Shantanu Das · Sebastien Tixeuil (Eds.)

LNCS 10641

# Structural Information and Communication Complexity

24th International Colloquium, SIROCCO 2017
Porquerolles, France, June 19–22, 2017
Revised Selected Papers

Springer

# Lecture Notes in Computer Science 10641

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

More information about this series at http://www.springer.com/series/7407

Shantanu Das · Sebastien Tixeuil (Eds.)

# Structural Information and Communication Complexity

## 24th International Colloquium, SIROCCO 2017
## Porquerolles, France, June 19–22, 2017
## Revised Selected Papers

Springer

*Editors*
Shantanu Das
LIF
Aix-Marseille University
Marseille Cedex 9
France

Sebastien Tixeuil
LIP6
Université Pierre et Marie Curie - Paris 6
Paris
France

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-72049-4 ISBN 978-3-319-72050-0 (eBook)
https://doi.org/10.1007/978-3-319-72050-0

Library of Congress Control Number: 2017960878

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues



Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

This volume contains the papers presented at the 24th International Colloquium on Structural Information and Communication Complexity: SIROCCO 2017, held during June 19–21, 2017, at Porquerolles, France. SIROCCO is devoted to the study of the interplay between structural knowledge, communication, and computing in decentralized systems of multiple communicating entities. Special emphasis is given to innovative approaches leading to better understanding of the relationship between computing and communication. The typical areas of interest include distributed computing, communication networks, game theory, parallel computing, social networks, mobile computing (including autonomous robots), peer-to-peer systems, and communication complexity.

This year we received 41 submissions in response to the call for papers and the Program Committee decided to accept 21 papers after a careful review and in-depth discussions. Each submission was reviewed by at least three reviewers; we had a total of 20 Program Committee members supported by 42 additional reviewers. The article "Leader Election in SINR Model with Arbitrary Power Control," by Magnus M. Halldorsson, Stephan Holzer, and Evangelia Anna Markatou received the Best Paper Award at SIROCCO 2017. Selected papers from the colloquium were invited to a special issue of the *Theoretical Computer Science* journal.

In addition to the regular contributed talks, the conference program included invited talks by Faith Ellen, Christian Scheideler, and Christoph Lenzen, and the award lecture by Shmuel Zaks, who received the 2017 SIROCCO Prize for Innovation in Distributed Computing. Short abstracts of all invited lectures, as well as a laudatio summarizing the numerous important innovative contributions of the award recipient Shmuel Zaks, are included in this volume.

We would like to thank all authors for their high-quality submissions and all speakers for their excellent talks at the conference. We are specially grateful to the Program Committee members and all additional reviewers who worked on a tight schedule to deliver an excellent conference program. The preparation of this event was guided by the SIROCCO Steering Committee, headed by Andrzej Pelc; we thank them for their help and support. The local organization of the conference was possible due to the efforts of the members of the Organizing Committee who were supported by the staff from the Laboratoire d'Informatique Fondamentale (LIF) at the Aix-Marseille University. The organizers of SIROCCO 2017 would like to acknowledge the financial support of our sponsors: LIF—Laboratory of Theoretical Computer Science, LabEx-Archimede Center of excellence, FRIIAM—Federation for research in Computer Science and Interactions, CNRS France, and Aix-Marseille University. Springer

not only helped with the publication of these proceedings but also sponsored the best paper award. The EasyChair system was used to handle the submission of papers, manage the review process, and generate these proceedings.

August 2017

Shantanu Das
Sebastien Tixeuil

# Organization

## Program Committee

| | |
|---|---|
| Dan Alistarh | ETH Zurich, Switzerland |
| Silvia Bonomi | Sapienza Università di Roma, Italy |
| Shantanu Das | Aix-Marseille University, France |
| Yann Disser | TU Darmstadt, Germany |
| Guy Even | Tel Aviv University, Israel |
| Antonio Fernandez Anta | IMDEA Networks Institute, Spain |
| Leszek Gasiniec | University of Liverpool, UK |
| Rastislav Kralovic | Comenius University, Slovakia |
| Danny Krizanc | Wesleyan University, USA |
| Bernard Mans | Macquarie University, Australia |
| Euripides Markou | University of Thessaly, Greece |
| Jaroslav Opatrny | Concordia University, Canada |
| Rotem Oshman | Tel Aviv University, Israel |
| Marina Papatriantafilou | Chalmers University of Technology, Sweden |
| Joseph Peters | Simon Fraser University, Canada |
| Guiseppe Prencipe | Università di Pisa, Italy |
| Stefan Schmid | Aalborg University, Denmark and TU Berlin, Germany |
| Sebastien Tixeuil | LIP6, UPMC Paris 6, France |
| Koichi Wada | Hosei University, Japan |
| Yukiko Yamauchi | Kyushu University, Japan |

## Steering Committee

| | |
|---|---|
| Magnus Halldorsson | Reykjavik University, Iceland |
| Andrzej Pelc | Université du Quebec en Outaouais, Canada |
| Nicola Santoro | Carleton University, Canada |
| Christian Scheideler | University of Paderborn, Germany |
| Jukka Suomela | Aalto University, Finland |

## Organizing Committee

| | |
|---|---|
| Evangelos Bampas | LIF, CNRS and Aix-Marseille University, France |
| Jérémie Chalopin | LIF, CNRS and Aix-Marseille University, France |
| Shantanu Das | LIF, CNRS and Aix-Marseille University, France |
| Emmanuel Godard | LIF, CNRS and Aix-Marseille University, France |
| Damien Imbs | LIF, CNRS and Aix-Marseille University, France |
| Christina Karousatou | LIF, CNRS and Aix-Marseille University, France |
| Arnaud Labourel | LIF, CNRS and Aix-Marseille University, France |

## Additional Reviewers

Abeliuk, Andres
Bampas, Evangelos
Bonnet, François
Bramas, Quentin
Chalopin, Jérémie
Chitnis, Rajesh
Dereniowski, Dariusz
Di Luna, Giuseppe Antonio
Dubois, Swan
Défago, Xavier
Feuilloley, Laurent
Förster, Klaus-Tycho
Gavoille, Cyril
Hackfeld, Jan
Hopp, Alexander Vincent
Izumi, Taisuke
Karousatou, Christina
Katayama, Yoshiaki
Keramatian, Amir
Kranakis, Evangelos
Najdataei, Hannaneh
Nanongkai, Danupon
Naves, Guyslain
Nicolaou, Nicolas
Nikolakopoulos, Yiannis
Ooshita, Fukuhito
Pagli, Linda
Panagiotou, Konstantinos
Patt-Shamir, Boaz
Peleg, David
Radzik, Tomasz
Sakavalas, Dimitris
Salem, Iosif
Savic, Vladimir
Schewior, Kevin
Schlöter, Miriam
Shibata, Masahiro
Sorella, Mara
Tudor, Valentin
Uznański, Przemysław
Viglietta, Giovanni
Yu, Dongxiao

# Laudatio

It is a pleasure to award the 2017 SIROCCO Prize for Innovation in Distributed Computing to Shmuel Zaks. Shmuel's contributions span an impressive range of research areas in computer science and discrete mathematics, including classic distributed computing (leader election, combinatorial and graph problems, complexity, impossibility, compact routing, self-stabilization, and more) and networking. Shmuel's work on networking has been performed during the past three decades; the first half of this period was devoted to ATM networks, and the second to optical networks.

The prize is awarded for these lifetime achievements, but especially for his pioneering research on algorithmic aspects of optical networks. In his seminal studies Shmuel formulated new problems and identified new research directions. The problems under investigation deal with a variety of aspects of optimization of the switching cost in the network, measured by the use of ADMs (ADD-DROP Multiplexers) and regenerators. Shmuel's studies deal with all algorithmic aspects of optimization problems that stem from optical networks, including the design and analysis of algorithms (e.g., approximation algorithms and on-line algorithms), complexity, parameterized complexity, and inapproximability. Shmuel's work initiated systematic studies of a variety of problems where mostly heuristics and simulations were previously used.

Examples of areas in which Shmuel's contributions to algorithmic aspects of optical networks are the most important include: ADM minimization [1, 2], regenerator placement [3, 4], traffic grooming [5], and the flex-grid model [6], where a lightpath has to be assigned a number of colors, within a contiguous or a non-contiguous range.

The 2017 Award Committee[1]

| | |
|---|---|
| Paola Flocchini | University of Ottawa |
| Magnús M. Halldórsson | University of Reykjavik |
| Thomas Moscibroda | Microsoft Research |
| Andrzej Pelc (Chair) | Université du Québec en Outaouais |
| Christian Scheideler | University of Paderborn |

[1] We wish to thank the nominators for the nomination and for contributing significantly to this text.

## Selected Publications Related to Shmuel Zaks' Contribution

1. Tamar Eilam, Shlomo Moran and Shmuel Zaks, Lightpath Arrangement in Survivable Rings to Minimize the Switching Cost, IEEE Journal on Selected Areas in Communications (JSAC), special issue on WDM-based network architectures, 20(1), January 2002, pp. 172–182.
2. Tamar Eilam, Shlomo Moran and Shmuel Zaks, Approximation Algorithms for Survivable Optical Networks, Proceedings of the 14th International Workshop on Distributed Algorithms (DISC), Toledo, Spain, October 2000, pp. 104–118.
3. Michele Flammini, Alberto Marchetti Spaccamela, Gianpiero Monaco, Luca Moscardelli and Shmuel Zaks, On the Complexity of Placement of Regenerators in Optical Networks, Proc. 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Calgary, Canada, August 2009, pp. 154–162.
4. George B. Mertzios, Ignasi Sau, Mordechai Shalom and Shmuel Zaks, Placing Regenerators in Optical Networks to Satisfy Multiple Sets of Requests, Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP), Bordeaux, France, July 2010, pp. 333–344. Best paper award.
5. Ignasi Sau, Mordechai Shalom and Shmuel Zaks, Traffic Grooming in Star Networks via Matching Techniques, Proc. 17th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Nesin Mathematics Village, Şirince, Turkey, June 2010, pp. 41–56.
6. Mordechai Shalom, Prudence W.H. Wong and Shmuel Zaks, Profit Maximization in Flex-Grid All-Optical Networks, Proc. 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Ischia, Italy, July 2013, pp. 249–260.

# Invited Presentations

# Online and Approximation Algorithms for Optical Networks and Scheduling

Shmuel Zaks

Department of Computer Science, Technion, Haifa, Israel
zaks@cs.technion.ac.il

**Abstract.** We discuss two fundamental problems that stem from optical networks models: minimizing the number of Add-Drop Multiplexers (ADMs) and regenerators. When also traffic grooming is allowed, then for a path topology network these problems can be interpreted as scheduling problems. We discuss various algorithmic aspects of several such optimization problems in offline and online settings, following [3, 4, 7, 8].

**Note:** This is a brief summary of the talk presented in Sirocco 2017. As such and due to space limitation, it contains neither a comprehensive reference list, nor a literature survey, nor extensions that stem from either optical networks, or scheduling theory, or the theory of algorithms; these all can be found, to a great extent, in the papers listed in the reference list.

## 1 Introduction – Optical Networks and Problem Definition

**Background:** Optical wavelength-division multiplexing (WDM) is the most promising technology today that enables us to deal with the enormous growth of traffic in communication networks, like the Internet. A communication between a pair of nodes is done via a *lightpath*, which is assigned a certain wavelength. In graph-theoretic terms, a lightpath is associated with a simple path in the network and a wavelength with a color assigned to it. We concentrate on the hardware cost, in terms of ADMs and regenerators.

**ADMs:** Each lightpath uses two Add-Drop Multiplexers (ADMs), one at each endpoint. If two adjacent lightpaths, i.e. lightpaths sharing a common endpoint, are assigned the same wavelength, then they can use the same ADM, provided their concatenation is a simple path. An ADM may be shared by at most two lightpaths. The total cost considered is the total number of ADMs. We thus want to color the lightpaths while minimizing the total number of ADMs, which we term the *minADM* problem. For a detailed technical explanation see [5].

**Grooming:** The problem of grooming is central in studies of optical networks. In graph-theoretic terms, we are given $g > 0$ and a set of simple paths, and we need to assign colors to the paths so that the union of all paths that get a particular color is a collection of disjoint simple paths in the graph, and such that at most $g$ of them ($g$ is

termed the grooming factor) can share the same edge. When the network topology is a path, this corresponds to scheduling problems, where the path is interpreted as the time axis, a lightpath as a job to be processed within the time interval represented by its endpoints, and the grooming factor as the number of jobs that one machine can process simultaneously. For a detailed technical explanation see [6].

**Regenerators:** The energy of the signal along a lightpath decreases and thus amplifiers are used every fixed distance. Yet, as the amplifiers introduce noise into the signal there is a need to place regenerators in nodes in the network along the lightpath every at most $d$ nodes, for a given parameter $d$. We thus want to color the lightpaths while minimizing the total number of regenerators, which we term the *minREG* problem. As each regenerator can serve only one lightpath, the basic problem is clearly optimized by placing a regenerator after exactly $d$ nodes on each lightpath separately. It becomes very difficult in two scenarios. The first one is when traffic grooming is allowed. The second one is when we are given $p$ sets of lightpaths, and we want to place regenerators so as to satisfy each of these sets separately. A theoretical model was suggested in [2], where also a detailed technical explanation can be found.

**Approximation Algorithms:** Given an NP-hard minimization problem $\Pi$, we say that a polynomial-time algorithm $\mathcal{A}$ is an $\alpha$-approximation algorithm, $\alpha \geq 1$, if for any problem instance $\mathcal{A}$ finds a feasible solution with cost at most $\alpha$ times the cost of an optimal solution. The class APX (Approximable) contains all NP-hard problems that can be approximated within a constant factor, and its subclass PTAS (Polynomial Time Approximation Scheme) contains the problems that can be approximated in polynomial time within a factor $1+\varepsilon$ for *any* fixed $\varepsilon > 0$; assuming $P \neq NP$ this subclass is proper. An APX-hardness result for a problem implies the non-existence of a PTAS (see [9]).

**Online Algorithms:** An online minimization algorithm is said to be $c$-competitive if for any input, it produces a solution that is at most $c$ times that used by an optimal offline algorithm (see [1]).

## 2 Discussion

Four results are presented, as follows:

1. Following [8] we discuss the online version of the *minADM* problem. This corresponds to scheduling jobs to machines, where the cost is associated with opening a machine, closing a machine, or moving a machine from one job to another. We show a competitive ratio of $\frac{7}{4}$ for any network topology, including rings of size at least four, $\frac{5}{3}$ for a triangle network, and $\frac{3}{2}$ for a path topology, and show that these results are best possible.
2. Following [4] we discuss online algorithm for the *minADM* problem when grooming is allowed. The cost of a coloring is the number of ADMs; in case $g$ lightpaths of the same wavelength enter through the same edge to one node, they can all use the same ADM (thus saving $g-1$ ADMs). This problem is NP-complete even for $g = 1$. Exact solutions are known for some specific cases, and approximation algorithms for certain topologies exist for $g = 1$. We discuss an

approximation algorithm for this problem. For every value of $g$ the running time of the algorithm is polynomial in the input size, and its approximation ratio for a wide variety of network topologies — including the ring topology — is shown to be $2lng + o(lng)$. This is the first approximation algorithm for the grooming problem with a general grooming factor $g$.

3. Following [3] we discuss the *minREG* problem, when grooming is allowed. Recall that in this setting a regenerator can serve up to $g$ lightpaths, while in scheduling this corresponds to the case in which up to $g$ jobs can be processed simultaneously by a single machine and the goal is to assign the jobs to machines so that the total busy time is minimized. The problem is NP-hard already for $g = 2$. We discuss an algorithm whose competitive ratio is between 3 and 4.
4. Following [7], we discuss the *minREG* problem, where for given $d > 0$ and $p > 0$ sets of lightpaths, we need to place regenerators so that, for each of the $p$ sets separately, there is a regenerator on each of its lightpaths after at most $d$ nodes. In scheduling this corresponds to the case where we have $p$ sets of jobs, each needs a service within $d$ time units, and we need to place a smallest number of machines such that for each set $A$, each of the jobs in $A$ is satisfied as desired. While this problem can be easily solved when $d = 1$ or $p = 1$, we discuss that for any fixed $d, p \geq 2$, it does not admit a PTAS, even if G has maximum degree at most 3 and the lightpaths have $O(d)$ lengths.

## References


1. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
2. Flammini, M., Marchetti-Spaccamela, A., Monaco, G., Moscardelli, L., Zaks, S.: On the complexity of the regenerator placement problem in optical networks. IEEE/ACM Trans. Netw. **19**(2), 498–511 (2011)
3. Flammini, M., Monaco, G., Moscardelli, L., Shachnai, H., Shalom, M., Tamir, T., Zaks, S.: Minimizing total busy time in parallel scheduling with application to optical networks. Theor. Comput. Sci. **411**(40–42), 3553–3562 (2010)
4. Flammini, M., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the traffic grooming problem. J. Discrete Algorithms **6**(3), 472–479 (2008)
5. Gerstel, O., Lin, P., Sasaki, G.: Wavelength assignment in a wdm ring to minimize cost of embedded sonet rings. In: INFOCOM 1998, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (1998)
6. Gerstel, O., Ramaswami, R., Sasaki, G.: Cost effective traffic grooming in wdm rings. In: INFOCOM 1998, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (1998)
7. Mertzios, G.B., Sau, I., Shalom, M., Zaks, S.: Placing regenerators in optical networks to satisfy multiple sets of requests. IEEE Trans. Netw. **20**(6), 1870–1879 (2012)
8. Shalom, M., Wong, P.W., Zaks, S.: Optimal on-line colorings for minimizing the number of adms in optical networks. J. of Discrete Algorithms **8**(2), 174–188 (2010)
9. Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press (2011)

# Ignorance is Bliss (for Proving Impossibility)

Faith Ellen

Department of Computer Science, University of Toronto, Canada
faith@cs.toronto.edu

This talk surveys techniques for proving uncomputability results and lower bounds in message passing models. An uncomputability result tells us that a certain problem cannot be solved in a particular model and a lower bound tells us that a certain problem cannot be solved in a particular model when insufficient resources are available. All of these impossibility results hinge on the processes' continued ignorance of input values, network parameters, or one another's states. Ignorance can also arise from asynchrony and faults.

The first problem considered is leader election. With anonymous processes, a *symmetry argument* shows that this problem cannot be solved deterministically, even if the system is synchronous, there are no faults, message sizes are unlimited, and all processes know that the network is a cycle of a particular length [A90]. Moreover, there is no randomized algorithm for leader election if all processes know that the network is a cycle, but do not know its length [A90]. In asynchronous models where processes have distinct identifiers, if at least half the processes can crash, then leader election is shown to be uncomputable using a *partition argument* [BT85]. If fewer processes can crash, then an *adversary argument* is presented which proves that the worst case expected message complexity is quadratic in the number of processes in the network [AGV15].

A simple *reduction* from leader election shows that the consensus problem is unsolvable in an asynchronous message passing system if at least half the processes can crash. But unlike the case for leader election, consensus remains unsolvable when only one process might crash. The *valency argument* was introduced to prove this result [FLP85]. One part of the proof is a *chain argument*, which establishes the existence of a multivalent initial configuration in any consensus algorithm. The other part is an *adversary argument*, showing that from any multivalent configuration and any step which can be performed in that configuration, there is a finite execution starting from that configuration and ending with that step which results in a multivalent configuration.

For the remainder of the talk, we consider synchronous networks of nonfaulty processes with distinct identifiers. When there is no bound on the size of the messages processes can send to their neighbours, an *information theoretic argument* is used to prove that, in a cycle of even length, the number of rounds needed to colour the processes with two colours (so that no two adjacent processes have the same colour) is linear in the length of the cycle [L92]. A simple *distance argument* shows that the number of rounds necessary to determine the diameter of a network is equal to its diameter.

Finally, in the CONGEST model, where each message is restricted to contain at most $B$ bits, a *communication complexity argument* is presented which proves that any algorithm for determining the diameter of a network with $n$ processes requires $\Omega(n/B)$ rounds, even if all processes know that the diameter is either two or three [FHW12]. The idea is to reduce the set disjointness problem, which is known to require a linear number of bits of communication between two players, to determining the diameter of a network from a specially designed class.

Many additional examples of impossibility proofs can be found in two survey papers [L89, FR03] and a recent book [AE14].

## References

[AGV15] Alistarh, D., Gelashvili, R., Vladu, A.: How to elect a leader faster than a tournament. In: Proceedings of the 34th Annual ACM Symposium on Principles of Distributed Computing, PODC 2015, pp. 365–374 (2015)

[A90] Angluin, D.: Local and global properties in networks of processors (Extended Abstract). In: Proceedings of the 12th Annual ACM Symposium on Theory of Computing, STOC 1980, pp. 82–93 (1980)

[AE14] Attiya, H., Ellen, F.: Impossibility Results for Distributed Computing. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2014)

[BT85] Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. J. ACM **32** (4), 824–840 (1985)

[FR03] Fich, F.E., Ruppert, E.: Hundreds of impossibility results for distributed computing. Distrib. Comput. **16**(2–3), 121–163 (2003)

[FLP85] Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)

[FHW12] Frischknecht, S., Holzer, S., Wattenhofer, R.: Networks cannot compute their diameter in sublinear time. In: Proceedings of the 23rd Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 1150–1162 (2012)

[L92] Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)

[L89] Lynch, N.A.: A hundred impossibility proofs for distributed computing. In: Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing, PODC 1989, pp. 1–28 (1989)

# Amoebots and Beyond: Models and Approaches for Programmable Matter

Christian Scheideler

Department of Computer Science, Paderborn University, Germany
scheideler@upb.de

**Abstract.** "Programmable matter" is a term originally coined by Toffoli and Margolus in 1991 to refer to an ensemble of fine-grained computing elements arranged in space. Since then, there has been a significant amount of work on programmable matter across multiple disciplines, including physics (e.g., crystals and complex fluids), chemistry (e.g., metamaterials and shape-changing molecules), bioengineering (DNA self-assembly and cell engineering), and robotics (modular robotics and nano robotics). So one can imagine that someday one can tailor-make programmable devices at nano-scale. However, before producing such devices, basic research is needed on the right primitives for these devices so that they can be used effectively in applications relevant for programmable matter. I will present the Amoebot model, which can be used effectively for typical applications like shape formation and coating, and discuss possible extensions of that model.

**Keywords:** Models · Self-organizing systems · Programmable matter

## 1 Introduction

Programmable matter promises to be a highly innovative technology with many interesting applications, ranging from minimal invasive surgery and adaptive materials to terraforming. Various models for programmable matter have already been proposed in different communities using approaches that differ from each other but also share some common ideas. For example, in the computational geometry community, *moteins*, whose designs are motivated by the folding behaviors of proteins, have been proposed to create complex shapes at the nanoscale [3]. Moteins consist of strings of very simple robotic modules that can fold into any volumetric shape. Additionally, motivated by computational origami, reconfigurable robots based on more complex folding primitives have been proposed. In the distributed computing community, we proposed *Amoebots* as a model for programmable matter that can adaptively form shapes and coat objects [5]. Related approaches can be found, e.g., in [8, 11]. In the DNA computing community, a number of DNA self-assembly models have been proposed. In the most basic model, the *abstract tile-assembly model* (aTAM), there are quadratic tiles with a specific glue on each side [9]. Equal glues have specific connection strengths and may bind together. Standard problems are to minimize the tile complexity (i.e., the number of different tile types) in order to form certain shapes and

to intrinsically perform computations which guide the assembly process. Whereas in the aTAM only individual tiles can be attached to an existing assembly, in more complex hierarchical assembly models, partial assemblies can also bind to each other (e.g., [4]). Beyond these passive self-assembly models, active self-assembly models based on molecular motors have been proposed, like the *nubot* model [12]. Finally, in the swarm robotics community, various prototypes of modular robots such as AMAS [10] and Mori [1] have been built in order to form complex robotic systems. In contrast to the previously mentioned models, these modular robots are computationally powerful devices. Much simpler robots have been proposed in the micro/nanorobotics field, including DNA machines, synthetic bacteria, nanoparticles, and magnetic materials, but these devices are designed and only useful for very specific tasks. More universal approaches are still under investigation.

## 2 The Amoebot Model

The Amoebots may form any subgraph of the infinite triangular grid $G_{\text{eqt}} = (V, E)$, where $V$ represents all possible positions the Amoebots can occupy relative to their structure, and $E$ represents all possible atomic movements an Amoebot can perform as well as all places where neighboring Amoebots can bond to each other. Figure 1(a) illustrates the standard planar embedding of $G_{\text{eqt}}$. We chose the triangular grid because in contrast to the hexagonal and square grids it guarantees that any Amoebot on the boundary of the Amoebot structure can move to an unoccupied neighboring node where it is able to bond again to a neighboring Amoebot.

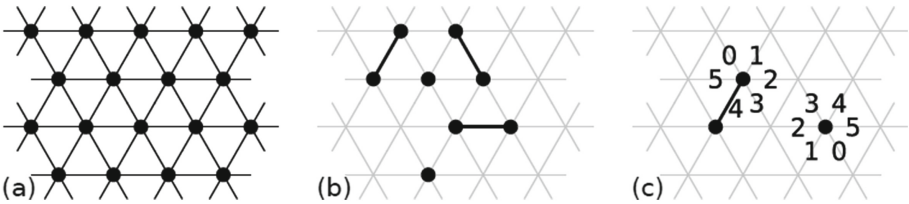



**Fig. 1.** (a) shows a section of $G_{\text{eqt}}$. Nodes of $G_{\text{eqt}}$ are shown as black circles. (b) shows five Amoebots on $G_{\text{eqt}}$. The underlying graph $G_{\text{eqt}}$ is depicted as a gray mesh. A Amoebot occupying a single node is depicted as a black circle, and a Amoebot occupying two nodes is depicted as two black circles connected by an edge. (c) depicts two Amoebots occupying two non-adjacent positions on $G_{\text{eqt}}$. The Amoebots have different offsets for their head port labelings.

Each Amoebot occupies either a single node or a pair of adjacent nodes in $G_{\text{eqt}}$, and every node can be occupied by at most one Amoebot. Two Amoebots occupying adjacent nodes are connected by a *bond*, and we refer to such Amoebots as *neighbors*. The bonds not only ensure that the Amoebot system forms a connected structure, but are also used for exchanging information.

Amoebots move through *expansions* and *contractions*: if an Amoebot occupies one node (i.e., it is *contracted*), it can expand to an unoccupied adjacent node to occupy two nodes. If an Amoebot occupies two nodes (i.e., it is *expanded*), it can contract to one of these nodes to occupy only a single node. Figure 1(b) illustrates a set of Amoebots (some contracted, some expanded) on the underlying graph $G_{\text{eqt}}$. We chose these kinds of movements since they allow Amoebots to stay connected while they move and it is easy to resolve movement conflicts by retracting to the contracted state. A *handover* allows two Amoebots to stay connected as they move. Two scenarios are possible: (1) a contracted Amoebot $p$ can "push" a neighboring expanded Amoebot $q$ and expand into a node previously occupied by $q$, forcing $q$ to contract, or (2) an expanded Amoebot $p$ can "pull" a neighboring contracted Amoebot $q$ to a node $v$ it occupies, causing $q$ to expand into $v$ and allowing $p$ to contract.

Amoebots are *anonymous*; they have no unique identifiers. Instead, each Amoebot has a collection of *ports* — one for each edge incident to the node(s) the Amoebot occupies — that have unique labels from the Amoebot's local perspective. We assume that the Amoebots have a common *chirality* (i.e., a shared notion of *clockwise direction*), which allows each Amoebot to label its ports in clockwise order. This is justified by the assumption that Amoebots can only bond with the same face up, which is also a common assumption in DNA computing. However, Amoebots do not have a common sense of global orientation and may have different offsets for their port labels, as in Fig. 1(c). Whenever Amoebots $p$ and $q$ share a bond, we assume that $p$ knows the label of $q$'s port it bonds to and whether $q$'s port belongs to the head or tail of $q$.

Each Amoebot has a constant-size local memory that can be read and written to by any neighboring Amoebot. Amoebots exchange information with their neighbors by simply writing into their memory. An Amoebot always knows whether it is contracted or expanded, and we assume that this information is also available to its neighbors (by publishing it in its local memory). Due to the constant-size memory constraint, Amoebots neither know the total number of Amoebots in the system nor any estimate of this number.

We assume the standard asynchronous model, where the Amoebot system progresses through a sequence of *Amoebot activations*; i.e., only one Amoebot is active at a time. Whenever a Amoebot is activated, it can perform an arbitrary bounded amount of computation involving its local memory and the memories of its neighbors and can perform at most one movement. A classical result under this model is that for any asynchronous concurrent execution of atomic Amoebot activations, there exists a sequential ordering of the activations which produces the same end configuration, provided conflicts which arise from the concurrent execution are resolved. We define an *asynchronous round* to be over once each Amoebot has been activated at least once. For more details, we refer to [6].

## 3 Future Developments

The Amoebot approach is currently explored in two further directions: One of these directions focuses on Amoebots that can sense and bond to other Amoebots but cannot

exchange information. Nevertheless, certain problems like the compression problem can be solved [2]. Another approach focuses on the problem of rearranging tiles into a specific shape using just a single Amoebot [7]. Many other variants are worth exploring since issues like fault tolerance, energy supply and consumption, and 3D structures have not been considered yet. So there is a large potential for future research in this area.

## References


1. Belke, C., Paik, J.: Mori: a modular origami robot. IEEE/ASME Trans. Mechatronics (2017, to be published)
2. Cannon, S., Daymude, J., Randall, D., Richa, A.: A markov chain algorithm for compression in self-organizing particle systems. In: Proceedings of the 35th ACM Symp. on Principles of Distributed Computing, PODC 2016, pp. 279–288 (2016)
3. Cheung, K.C., Demaine, E., Bachrach, J.R., Griffith, S.: Programmable assembly with universally foldable strings (moteins). IEEE Trans. Robot. **27**(4), 718–729 (2011)
4. Demaine, E., Fekete, S., Scheffer, C., Schmidt, A.: New geometric algorithms for fully connected staged self-assembly. Theoret. Comput. Sci. **671**, 4–18 (2017)
5. Derakhshandeh, Z., Dolev, S., Gmyr, R., Richa, A., Scheideler, C., Strothmann, T.: Brief announcement: Amoebot - a new model for programmable matter. In: Proceedings of the 26th ACM Symp. on Parallelism in Algorithms and Architectures, SPAA 2014, pp. 220–222 (2014)
6. Derakhshandeh, Z., Gmyr, R., Strothmann, T., Bazzi, R., Richa, A., Scheideler, C.: Leader election and shape formation with self-organizing programmable matter. In: Proceedings of the 21st International Conference on DNA Computing and Molecular Programming, DNA 2015, pp. 117–132 (2015)
7. Gmyr, R., Kostitsyna, I., Kuhn, F., Scheideler, C., Strothmann, T.: Forming tile shapes with a single robot. In: European Workshop on Computational Geometry, EuroCG 2017 (2017)
8. Hurtado, F., Molina, E., Ramaswami, S., Sacristán, V.: Distributed reconfiguration of 2d lattice-based modular robotic systems. Auton. Robots **38**, 383–413 (2015)
9. Rothemund, P., Winfree, E.: The program-size complexity of self-assembled squares. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, STOC 2000, pp. 459–468 (2000)
10. Terada, Y., Murata, S.: Automatic modular assembly system and its distributed control. Int. J. Robot. Res. **27**(3–4), 445–462 (2008)
11. Walter, J., Welch, J., Amato, N.: Distributed reconfiguration of metamorphic robot chains. Distrib. Comput. **17**(2), 171–189 (2004)
12. Woods, D., Chen, H.-L., Goodfriend, S., Dabby, N., Winfree, E., Yin, P.: Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In: Innovations in Theoretical Computer Science, ITCS 2016, pp. 353–354 (2013)

# The Many Faces of Clock Synchronization

Christoph Lenzen

Max Planck Institute for Informatics, Saarland Informatics Campus

**Abstract.** Reliable and scalable clocking of hardware systems requires fault-tolerant distributed clocking methods. A brief and incomplete overview of contemporary challenges on this front is given here.

Today's hardware has a number of characteristics that renders a multi-core system or even a single chip a distributed system. A modern chip comprises billions of transistors, which work in parallel, and operates at gigahertz speeds. This entails that the system must be robust to both transient and permanent faults. Typically, the operation is synchronized, i.e., the gates' transitions are *clocked.* This requires to painfully accurately distribute a clock signal throughout the chip, as already timing deviations in the order of tens of picoseconds violate the specification under which the chip is guaranteed to operate correctly. The traditional solution are clock trees, which spread the clock signal from a single source, e.g., a quartz oscillator, throughout the chip. However, clock trees suffer from limitations in scalability and introduce a single point of failure.

*Byzantine Fault-tolerant Clock Synchronization.* To tackle these issues, one may employ a *clock synchronization algorithm* to synchronize multiple clock sources, which then each drive a small, local clock tree to run (possibly redundant) subsystems. Referring to these *clock domains* as nodes, a classic distributed model for clock synchronization arises. We assume (i) a fully connected system of $n$ nodes, (ii) at most $f < n/3$ *Byzantine faults* (i.e., arbitrary behavior), (iii) for each node a local clock source of *bounded drift* (i.e., at all times running at a rate between 1 and $\vartheta > 1$), and (iv) *bounded delay,* i.e., each message takes between $d - \varepsilon$ and $d$ time to arrive (where computational delay is accounted for in $d$).

A classic algorithm in this model is due to Srikanth and Toueg [12], achieving a *skew* of $2d$, i.e., the maximum time difference between corresponding clock "ticks" of non-faulty nodes is $d + \varepsilon$. It has been successfully implemented in hardware [5]. This solution has two crucial downsides. First, the skew lower bound of $\Omega(\varepsilon)$ [10] is not matched, and second the algorithm is not *self-stabilizing,* i.e., correct operation is not re-established after (an unbounded number of) transient faults.

*Pulse Generation.* With self-stabilization as an additional requirement, the problem has been studied under the name Byzantine fault-tolerant self-stabilizing *pulse generation.* Due to the substantial challenge this imposes, many algorithms need to communicate large amounts of information. Thus, the amounts of bandwidth per node (number of broadcasted bits per time unit) becomes an additional quality measure. The currently best algorithms result from a recent framework for reducing the problem to

synchronous consensus was presented [9], resulting in a solution with stabilization time and bandwidth polylog$f$. This reduction "translates" the running time of the consensus algorithm to stabilization time and its message size to bandwidth, with at most a factor $O(\log f)$ increase in each.

**Open Problem 1** *Is pulse synchronization at least as hard as consensus?*

*Metastability.* The second issue of the Srikanth-Toueg algorithm, the potentially large gap between the achieved skew and the lower bound of $\Omega(\varepsilon)$, has been addressed by another classic synchronization algorithm, due to Lundelius Welch and Lynch [13]. We implemented a variant of this algorithm in hardware [7], with surprisingly good results. However, the achieved precision is still by roughly an order of magnitude worse than that of a clock tree, necessitating further improvements. Doing so requires to overcome a fundamental obstacle: *metastability.* Metastability is an unstable equilibrium state of a bistable element (such as a register or flip-flop) that may occur when input signals are "unclean" and violate timing constraints. Marino proved [11] that the possibility of metastability cannot be avoided when measuring the skew between different clocks, which naturally is necessary for any clock synchronization algorithm.

Usually, metastability is handled by simply waiting for the register, flip-flop, etc. to recover a stable state with sufficiently high probability. However, when trying to run the Lynch-Welch algorithm at sufficiently high frequency to achieve the desired skew bound, there is insufficient time for this solution. Another issue is that a faulty node could provide out-of-spec input signals to a correct node, possibly "infecting" it with metastability.

We introduced an approach based on Kleene logic [4] modelling metastability in a worst-case fashion. It represents "unclean" signals (intermediate voltages between logical 0 and 1, oscillations, late transitions, etc.) by a third symbol $M$ and extends the gate function $f : \{0,1\}^n \to \{0,1\}$ of a basic gate to its *metastable closure* $f_M : \{0,1,M\}^n \to \{0,1,M\}$. Defining for $x \in \{0,1,M\}^n$ that $Res(x) := \{x' \in \{0,1\}^n \,|\, x'_i \neq x_i \Rightarrow x_i \neq M\}$, i.e., the set of strings $x \in \{0,1\}^n$ obtained by treating $M$ as "wildcard," $f_M$ is given by $f_M(x) = b \in \{0,1\}$ iff $f(x') = b$ for all $x' \in Res(x)$ and $f_M(x) = M$ else. It was shown that for any Boolean function $f$, $f_M$ can be implemented using standard logic [4]. In particular, this makes it possible to implement the Lynch-Welch algorithm in a way that avoids metastability-induced faults deterministically. However, in general a circuit implementing $f_M$ may be of exponential size in $n$, even if one for $f$ is small.

**Open Problem 2** *For a Boolean function $f$, how large is a circuit that implements $f_M$ on all inputs with up to $k$ metastable bits?*

*Combining Self-stabilization and High Precision.* In [6], it is shown how to couple (a variant of) the Lynch-Welch algorithm with the solution from [2] to obtain a clock synchronization algorithm of skew $O(\varepsilon)$ and stabilization time $O(f)$. Unfortunately, this results in a significant increase in the circuitry for a single node, which is likely to increase the probability that an individual node fails. Therefore, it is desirable to find a

different coupling mechanism that ensures that the underlying pulse synchronization algorithm is only relied on during stabilization.

**Open Problem 3** *Find coupling mechanisms that enable self-stabilization of the Lynch-Welch algorithm without interfering with it after stabilization.*

*Distributing the Clock Signal.* All of the above solutions assume full connectivity, which is impractical across a whole system. Thus, a way of distributing the clock signal reliably using a low-degree topology is required. HEX [1, 8] provides a first stab at the problem, by ensuring both self-stabilization and resilience to one Byzantine fault in the neighborhood of each node. More generally, one may aim at tolerating $\Omega(\Delta)$ such *local* faults in degree-$\Delta$ networks. However, HEX is not sufficiently precise to be considered as general purpose clocking method.

**Open Problem 4** *Come up with competitive low-degree clock distribution networks that combine self-stabilization and tolerate local Byzantine faults.*

For further details on these challenges and others, please refer to our survey [3].

## References

1. Dolev, D., Fuegger, M., Lenzen, C., Perner, M., Schmid, U.: HEX: scaling honeycombs is easier than scaling clock trees. In: Proceedings Symposium on Parallelism in Algorithms and Architectures, SPAA 2013 (2013)
2. Dolev, D., Fuegger, M., Lenzen, C., Schmid, U.: Fault-tolerant algorithms for tick-generation in asynchronous logic: robust pulse generation. J. ACM **61**(5), 30:1–30:74 (2014)
3. Dolev, D., Fugger, M., Lenzen, C., Schmid, U., Steininger, A.: Fault-tolerant distributed systems in hardware. Bull. EATCS **116** (2015)
4. Friedrichs, S., Függer, M., Lenzen, C.: Metastability-containing circuits (2016). CoRR abs/1606.06570
5. Függer, M., Schmid, U.: Reconciling fault-tolerant distributed computing and systems-on-chip. Distrib. Comput. **24**(6), 323–355 (2012)
6. Khanchandani, P., Lenzen, C.: Self-stabilizing Byzantine clock synchronization with optimal precision. In: Proceedings of Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS 2016 (2016)
7. Kinali, A., Huemer, F., Lenzen, C.: Faul-tolerant clock synchronization with high precision. In: Symposium on VLSI, ISVLSI 2016 (2016)
8. Lenzen, C., Perner, M., Sigl, M., Schmid, U.: Byzantine self-stabilizing clock distribution with HEX: implementation, simulation, clock multiplication. In: Proceedings of Conference on Dependability, DEPEND 2013 (2013)
9. Lenzen, C., Rybicki, J.: Self-stabilising byzantine clock synchronisation is almost as easy as consensus. In: Proceedings of 31st Symposium on Distributed Computing, DISC 2017 (2017, to appear)
10. Lundelius, J., Lynch, N.: An upper and lower bound for clock synchronization. Inf. Comput. **62**(2–3), 190–204 (1984)

11. Marino, L.: General theory of metastable operation. IEEE Transac. Comput. **C-30**(2), 107–115 (1981)
12. Srikanth, T.K., Toueg, S.: Optimal clock synchronization. J. ACM **34**(3), 626–645 (1987)
13. Welch, J.L., Lynch, N.A.: A new fault-tolerant algorithm for clock synchronization. Inf. Comput. **77**(1), 1–36 (1988)

# Contents

**Probabilistic Algorithms**



**Computational Complexity**



**Dynamic Networks**

# Wireless Networks

# Leader Election in SINR Model with Arbitrary Power Control

Magnús M. Halldórsson[1], Stephan Holzer[2], and Evangelia Anna Markatou[2(✉)]

[1] ICE-TCS, School of Computer Science, Reykjavik University, Reykjavik, Iceland
mmh@ru.is

[2] TDS Group, Massachusetts Institute of Technology, Cambridge, USA
{holzer,markatou}@mit.edu

**Abstract.** In this article, we study the Leader Election Problem in the Signal-to-Interference-plus-Noise-Ratio (SINR) model where nodes can adjust their transmission power. We show that in this setting it is possible to solve the leader election problem in two communication rounds, with high probability. Previously, it was known that $\Omega(\log n)$ rounds were sufficient and necessary when using uniform power, where $n$ is the number of nodes in the network.

We then examine how much power control is needed to achieve fast leader election. We show that any 2-round leader election algorithm in the SINR model running correctly w.h.p. requires a power range $2^{\Omega(n)}$ even when $n$ is known. We match this with an algorithm that uses power range $2^{\tilde{O}(n)}$, when $n$ is known and $2^{\tilde{O}(n^{1.5})}$, when $n$ is not known. We also explore tradeoffs between time and power used, and show that to elect a leader in $t$ rounds, a power range $exp(n^{1/\Theta(t)})$ is sufficient and necessary.

**Keywords:** SINR · Leader election · Power control · Capture effect

## 1 Introduction

In this article we discuss what we can accomplish in a Signal-to-Interference-plus-Noise-Ratio (SINR) network using *power control*, the ability of nodes to transmit with variable transmission power, and the *capture effect*, a property of SINR networks, where a transmission can be successful while other transmissions within the communication range occur in the same round.

We study the leader election problem as a vehicle to explore this frontier. Leader election, the problem of determining a *unique leader* among the nodes in a network, is one of the oldest and most studied problems in distributed computing. It provides a strong form of breaking symmetry within radio networks in an

M. M. Halldórsson is supported by Icelandic Research Fund grants 152679-05 and 174484-05. Stephan Holzer is supported by AFOSR FA9550-13-1-0042. Evangelia Anna Markatou is supported by grants NSF CCF-1461559 and NSF CCF-0939370.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 3–14, 2017.
https://doi.org/10.1007/978-3-319-72050-0_1

initially unknown system, and is frequently used as a preliminary step in more complex communication tasks.

The leader election problem was originally introduced in the 1970s, with the publication of the ALOHA radio network paper [1]. In the following years, many variations of the leader election problem have been extensively studied under a variety of models and algorithmic constraints such as with collision detection in the multiple access channel model [12], no collision detection in the SINR model [7], or under a colored graph in the LOCAL model [6].

We treat the leader election problem in SINR networks, first studied by Gupta and Kumar [9] for algorithmic purposes. In the SINR model, nodes operate in synchronous rounds. In each round a node either broadcasts a message to its neighbors or listens. A node $v$ receives a message from node $u$ depending on the distance between $u$ and $v$, the transmission power of $u$, and the interference generated by other broadcasting nodes, as defined in Sect. 2.

The best solution known for this problem in an SINR network achieves $O(\log n)$ runtime with high probability (w.h.p.) using uniform transmission power [7]. In the classical radio network model, the leader election problem requires $\Theta(\log^2 n)$ rounds w.h.p. [11]. Fineman et al. [7] show that $O(\log n)$ rounds suffice to elect a leader in SINR networks without power control, and show that $\Omega(\log n)$ are also necessary when using uniform power. They suggest that improved bounds may be possible using power control. Indeed, we show that power control can provide the ultimate speedup.

**Our Contributions:** We present an algorithm that solves the leader election problem in two rounds w.h.p.. We also present a multi-round leader election algorithm that uses limited transmission power. Our work is complemented by nearly matching lower bounds on the transmission power range for both two round and multi-round leader election algorithms.

### 1.1 Related Work

The leader election problem was first studied with the publication of the ALOHA radio network in the 1970s [1], and plenty of work considering this problem was published in the following decade. Gallager [8] presents a good survey of early work on leader election. Starting in the 1990s there was an increased interest in the *radio network models* [4]. Under radio network models concurrent transmissions are lost due to collisions, and nodes do not know whether or not their message was successfully received. In this model, the leader election problem can be solved in $\Theta(\log^2 n)$ rounds w.h.p. [11] where $n$ is the number of nodes in the network. This bound can be improved to $\Theta(\log n)$ w.h.p. assuming that nodes can detect collisions [11], and to $O(\log n_u)$ expected rounds assuming an upper bound $n_u$ of $n$ [2].

In the beginning of the new millennium came a renewed interest in *fading radio networks*, captured with the SINR model, which claim to capture the real behavior of systems better than previous models, as they take interference into account in a more realistic way. Moscibroda and Wattenhofer [10] showed that

algorithms on the fading radio networks model can achieve better runtimes than algorithms for the radio networks model on certain problems, as SINR allows for better spatial reuse.

In the SINR model the most efficient currently published leader election protocol is by Fineman et al. [7]. The authors present an algorithm that achieves $O(\log n + \log R)$ runtime w.h.p. in a single-hop network using uniform transmission power, where $n$ is the number of nodes and $R = O(poly(n))$ is the ratio between the longest and shortest link. Fineman et al. suggest that it may be possible to achieve better performance using power control. Indeed, for problems like link scheduling and connectivity, power control has been shown to give much better performance [10]. Power control has also been used in the SINR setting to solve the link scheduling problem while conserving energy, e.g. [3,5].

To our knowledge, there has been no published work using power control to optimize the runtime of the leader election problem, or examining the trade-offs between the required communication complexity and power range of a leader election algorithm.

## 2 Model and Problem Statement

Let $V$ be a set of $n$ nodes, deployed in a single-hop network, that represent wireless devices. Every node can communicate with any other node using transmission power $P$, in absence of interference from other nodes. Time is divided into synchronous rounds. In each round, a node $v$ can either transmit a message of size $O(\log n)$ with some power $P_v$, or listen. Node $v \in V$ can receive a message transmitted by node $u \in V$, iff $v$ is listening and

$$SINR(u, v, I) = \frac{\frac{P_u}{d(u,v)^\alpha}}{N + \sum_{w \in I} \frac{P_w}{d(w,v)^\alpha}} \geq \beta, \tag{1}$$

where $I$ is the set of other nodes transmitting simultaneously. $d(u, v)$ is the distance between nodes $v$ and $u$, and $\alpha, \beta, N$ are constants. Specifically, $\alpha$ is the path-loss exponent, $N$ is the non-zero ambient noise, and $\beta$ is a hardware-dependent minimum SINR threshold required for a successful message reception. Our algorithms work for any $\beta > 0$, while the lower bounds use $\beta \geq 2$.

In this paper, we consider the leader election problem.

*Problem 1 (Leader Election Problem).* Given $n$ nodes in a network, eventually elect exactly one node *(called the leader)*, with all nodes knowing whether or not they were elected to be the leader.

We denote by $R$ the ratio of the longest to shortest distance between any two nodes in the network. Similar to [7], we assume that $R$ is bounded by a polynomial in $n$, $R \leq n^c$, for some $c \in \mathbb{N}$. Let $\gamma$ be a constant such that $\gamma \geq \max(1, c\alpha + 1 + \log \beta)$. We assume that the nodes know or can infer (an upper bound on) $\gamma$.

The $\tilde{O}$-notation omits logarithmic factors. All logs are base 2. We consider that an event happens with high probability (w.h.p.) if it happens with probability greater than $1 - 1/n$.

We need the following version of Chernoff bounds.

**Theorem 1 (Chernoff Bound).** *Let $X_1, X_2, \ldots, X_n$ be independent Bernoulli random variables and $X = \sum_{i=1}^{n} X_i$. Then,* $\Pr[X \geq 2 \cdot \mathbb{E}[X]] \leq 2^{-0.55\mathbb{E}[X]}$.

## 3 2-Round Leader Election Algorithm

In this section, we present a 2-round leader election algorithm. First, we give some key ideas behind our algorithm. Then, we present a 2-round leader election algorithm that requires no knowledge of $n$, followed by the analysis.

### 3.1 The Essence of Our Algorithm

Below we present a high level description of the key ideas behind our algorithm.

(i) **Geometric random variable:** The nodes use a geometric random variable $k$ to count the tails flipped in a sequence of coin flips before the first heads is flipped. This geometric random variable allows some nodes to approximate $n$ with no prior knowledge of the instance. More specifically, at least one and at most $8 \log n$ nodes flip a coin more than $\log n - \log \log n - 2$ times.

(ii) **Random $ID$s:** Each node chooses an $ID$ (identification number) randomly using $k$. The geometric random variable $k$ ensures that exactly one node $v$ holds the maximum $ID$, which allows node $v$ to break the symmetry of the network and stand out as the leader.

(iii) **The loudest node wins:** Each broadcasting node $v$ determines its transmission power by evaluating power function $f(ID_v) = P \cdot ID_v^{\gamma^{ID_v}}$ using its identification number, $ID_v$. Transmission power function $f$ ensures that all listening nodes receive a message exactly from the node with the largest $ID$, as long as that $ID$ is unique (see (ii)).

(iv) **Feedback:** In order to inform all nodes of the leader node $v$, we split the set of nodes $V$ into listeners and competitors. The competitors compete for the leader position during the first round. The listeners inform the competitors of the winner during the second round. Both rounds use the same protocol with different message contents.

In summary, a *geometric random variable* allows the nodes to approximate $n$ with no prior knowledge of the instance, *random $ID$s* ensure that the node $v$ with the maximum $ID$ stands out, arbitrary transmission power allows the *loudest node* $v$ to inform the other nodes it is the leader, and *feedback* makes sure that all nodes know who the leader node is.

### 3.2 Leader Election Algorithm

The algorithm proceeds as follows. Initially, each node $v$ flips a coin (a Bernoulli random variable) to determine its *role*, which is a *competitor* if heads are flipped, and *listener* if tails. It then computes a geometric random variable (r.v.) $k_v$, which counts the tails flipped in a sequence of coin flips before the first heads is flipped. The ID of the node, $ID_v$, is an integer selected uniformly at random from the range $[J, 2 \cdot J]$, where $J = g(k_v) := 2^{k_v} k_v^4$. Finally, the power $P_v$ that $v$ uses for broadcast is given by $f(ID_v) := P \cdot ID_v^{\gamma ID_v}$, where $P$ is the minimum power needed to reach all nodes in the network (overcoming the ambient noise).

During round 1, competitors transmit their ID using the assigned power $P_v$, which is to be received by the listeners. In round 2, the roles are reversed, as the listeners report back the ID of the purported leader that they received.

We shall argue that, with high probability, a unique competitor succeeds in transmitting to all the listeners, and a unique listener succeeds in reporting back to all the competitors. The leader is then that successful competitor.

**Algorithm 1.** 2-Round Leader Election Algorithm for node $v$

1: $Role_v$, a boolean $Bernoulli(\frac{1}{2})$ random variable {'competitor' if heads, 'listener' if tails}
2: $k_v$, a $Geometric(\frac{1}{2})$ random variable, $k_v \in \mathbb{Z}_{\geq 0}$
3: $ID_v$, chosen uniformly at random from $[J, 2 \cdot J)]$, where $J = g(k_v) := 2^{k_v} k_v^4$, $ID_v \in \mathbb{Z}_{\geq 0}$
4: $P_v$, the transmission power, $P_v = f(ID_v) := P \cdot (ID_v)^{\gamma ID_v}$, $P_v \in \mathbb{Z}_{\geq 0}$
5: $Leader_v$, a string denoting the identity of the leader, initially empty
6: **Round 1:**
7: **if** $Role_v$ = competitor **then**
8: Broadcast $ID_v$ using power $P_v$
9: **else**
10: Receive $Leader_v$
11: **Round 2:**
12: **if** $Role_v$ = competitor **then**
13: Receive $Leader_v$
14: **else**
15: Broadcast $Leader_v$ using power $P_v$

### 3.3 Analysis

We proceed by showing that the highest power used by a competitor is sufficient to overpower all the other competitors, ensuring that this competitor is heard by all the listeners. Identical arguments hold for the reporting back in round 2.

To this end, we first show that there is a competitor whose geometric r.v. is nearly $\log n$, and at most a logarithmic number of competitors have that large value. We then show that all the $O(\log n)$ IDs at the high end of the spectrum

are unique, i.e., selected by a single node. The difference in power used by nodes with different ID ensures that the competitor with highest ID will overpower all the other competitors and be heard by all the listeners.

**Lemma 1.** *Let $k_1 := \log n - \log\log n - 2$. For at least one and at most $8\log n$ competitors $v$ does it hold that $k_v \geq k_1$, with probability greater than $1 - \frac{1}{8n}$.*

*Proof.* Let $t = \lceil k_1 \rceil = \lceil \log n - \log\log n - 2 \rceil$. Let $A_v$ be the event that a given node $v$ is a competitor and has $k_v \geq t$. The probability of $A_v$ is $\Pr[A_v] = 2^{-1-t} = 2^{-1-\lceil k_1 \rceil}$. Thus,

$$\frac{2\log n}{n} = 2^{-1-k_1} \leq \Pr[A_v] \leq 2^{-k_1} = \frac{4\log n}{n}.$$

The probability that no node satisfies $A_v$ is then at most

$$\Pr\left[\bigwedge_v \overline{A_v}\right] \leq \left(1 - \frac{2\log n}{n}\right)^n \leq e^{-2\log n} \leq n^{-2.88} \leq \frac{1}{16n},$$

for $n$ sufficiently large, establishing the first part of the claim.

Let $X$ be the number of nodes $v$ for which $A_v$ holds. Then $2\log n \leq \mathbb{E}[X] \leq 4\log n$ and by Chernoff bound (Theorem 1),

$$\Pr[X \geq 8\log n] \leq \Pr[X \geq 2\mathbb{E}[X]] \leq 2^{-0.55\mathbb{E}[X]} < 2^{-2.2\log n} = n^{-2.2} \leq \frac{1}{16n},$$

for $n$ large enough. I.e., at most $8\log n$ nodes satisfy $A_v$, with probability greater than $1 - \frac{1}{16n}$.

Combined, with probability at least $1 - \frac{1}{8n}$, both of these claimed events hold.

The range from which the IDs are chosen is $[J, 2J]$, for $J \geq g(k_1)$, with high probability. Observe that $g(k_1) = 2^{k_1}k_1^4 \geq \frac{n\cdot\log^3 n}{8}$, for sufficiently large values of $n$.

**Lemma 2.** *A sole competitor receives the highest ID with probability greater than $1 - \frac{1}{8n}$, given that at least one node calculated $k_v \geq k_1$.*

*Proof.* The ranges of IDs assigned to nodes of different $k_v$ values are disjoint. The competitor receiving the highest ID will therefore necessarily be one with a highest $k_v$ value, which we denote by $K$. Let $Z$ be the set of competitors with $k_v = K \geq k_1 (= \log n - \log\log n - 2)$. By Lemma 1, $Z$ is non-empty and contains at most $8\log n$ nodes.

The probability that a given pair of nodes in $Z$ receive the same ID is inversely proportional to the range of IDs sampled from, or $1/J \leq \frac{1}{g(k_1)} \leq \frac{8}{n\cdot\log^3 n}$. The probability that some pair of nodes in $Z$ are assigned the same ID is then, by the union bound, at most

$$\frac{\binom{|Z|}{2}}{J} \leq \frac{(8\log n)^2}{\frac{n\cdot\log^3 n}{8}} = \frac{512}{n\log n} < \frac{1}{8n},$$

for large enough $n$. In particular, all nodes in $Z$ receive different IDs with probability greater than $1 - \frac{1}{8n}$.

The highest ID received, $ID_w$, is at least $g(k_1) \geq n$, for sufficiently large values of $n$.

**Lemma 3.** *If a sole competitor receives the highest ID, then its transmission is received by all the listeners.*

*Proof.* Let $w$ be the sole competitor with the highest $ID$. For any other competitor $v$ it then holds that

$$\frac{P_w}{P_v} \geq \frac{f(ID_w)}{f(ID_w - 1)} \geq ID_w^{\gamma} \geq n^{\gamma} \geq \beta n^{c\alpha+1}. \tag{2}$$

Let $u$ be a listener. We bound the noise and interference received by $u$ in terms of the signal $S_u := P_w/d(w,u)^{\alpha}$ it receives from $w$. Recall that $d(w,u) \leq R \cdot d(v,u) \leq n^c \cdot d(v,u)$, and thus $d(w,u)^{\alpha} \leq n^{c\alpha} \cdot d(v,u)^{\alpha}$, for any competitor $v$. Hence, applying (2), the interference received from a competitor $v$ is bounded by

$$I_v := \frac{P_v}{d(v,u)^{\alpha}} \leq \frac{P_w \cdot n^{c\alpha}}{\beta n^{c\alpha+1} \cdot d(w,u)^{\alpha}} = \frac{S_u}{\beta n}. \tag{3}$$

The definition of minimum power $P$ ensures that $\frac{P/d(w,u)^{\alpha}}{N} \geq \beta$. Thus, we can use (2) to bound the noise term by

$$N \leq \frac{P}{d(w,u)^{\alpha} \cdot \beta} \leq \frac{P_w}{d(w,u)^{\alpha} \cdot n^{\gamma} \cdot \beta} = \frac{S_u}{\beta n^{\gamma}} \leq \frac{S_u}{\beta n}. \tag{4}$$

Combining (3) and (4), we get that the SINR of $w$'s signal at receiver $u$ is bounded below by

$$\frac{S_u}{N + \sum_{v \in X} I_v} \geq \frac{\beta n}{1 + |X|} \geq \beta,$$

where $X$ is the set of competitors other than $w$. Thus, $w$ overpowers all other competitors at all the listeners.

**Theorem 2.** *The 2-round leader election algorithm terminates with all nodes agreeing on a common leader, w.h.p.*

*Proof.* Adding up the error probabilities of Lemmas 1 and 2, we find that a sole competitor $w$ receives the highest ID, with probability at least $1 - \frac{1}{4n}$. By Lemma 3, $w$ then successfully informs all the receivers. All three lemmas work identically for the reporting process in round 2. Hence, with probability at least $1 - \frac{1}{2n}$, the algorithm succeeds.

*Remark 1.* Leader election can be achieved in a single round if simultaneous transmission and reception is possible. Such *full-duplex* radios operate by subtracting the transmitted signal from the received one. While they are still rare, being hard to implement, such technology has been progressing significantly in recent years and may well become a commodity feature. With full-duplex, our arguments apply unchanged to the success of reception by the other competitors, thus succeeding after only a single round.

# 4 Range of Power Needed for a 2-Round Leader Election

Power control is the essential feature that allows our algorithms to work. That begs the question *how much* power control is needed?

We say that an algorithm uses a *power range* $X$ if the powers assigned fall in the range $[P, \ldots, X \cdot P]$. The basic question is then how the power range must grow as a function of $n$ for leader election to work correctly.

## 4.1 Upper Bound

**Theorem 3.** *Our 2-round leader election algorithm can be made to work correctly with a power range of* $2^{\tilde{O}(n^{1.5})}$*, w.h.p.*

*Proof.* The algorithm as is may select power assignments inducing a range of $2^{\tilde{O}(n^2)}$, since $k_v$ is no larger than $2\log n + 2$, with probability greater than $1 - \frac{1}{2n}$. However, if the range is bounded, we may assume that the nodes know the upper bound of the range, $P_{max}$. Thus, the algorithm would automatically truncate the power assigned to be at most $P_{max}$. We observe that this truncation can occur for at most one vertex, for the node with the highest ID to succeed. Namely, the probability that two or more nodes select a $k_v$ value greater than $1.5 \log n$ is at most

$$\binom{n}{2} 2^{-3\log n} \leq \frac{1}{2n}.$$

The bound on the maximum power now follows immediately.

If nodes know $n$, we can work with a smaller power range as follows: We can first sample the nodes with probability $\Theta(\log n/n)$, and have each selected node select ID uniformly at random from the range $[J, 2J]$, where $J = n\log^2 n$. The power used is $f(ID_v)$ as before, and the arguments are otherwise the same. This results in a power range of at most $2(n\log^2 n)^{n\log^2 n} = 2^{\tilde{O}(n)}$.

**Proposition 1.** *When nodes know* $n$*, a power range of* $2^{\tilde{O}(n)}$ *suffices.*

## 4.2 Lower Bound

We show that an exponential-size power range is actually necessary for any leader election protocol running in (at most) two rounds.

**Theorem 4.** *Every 2-round leader election algorithm in the SINR model running correctly w.h.p. requires a power range* $2^{\Omega(n)}$*. This holds even if the nodes know* $n$*, the number of nodes in the network, and if the nodes are located in a unit metric space (where all distances are equal).*

*Proof.* Consider $n$ nodes located in a unit metric. In the unit metric, either a single message is received by all the listeners or none of them hear anything (assuming $\beta \geq 1$). Since the nodes don't operate full-duplex, two rounds are

needed to inform the transmitting nodes of the winner, and the winner must be heard by all listeners in the first round.

We divide the available range of power into subranges, each within factor 2. Namely, if $P_{max}$ is the maximum power available, then the $i$-th highest subrange is $[P_{max}/2^i, P_{max}/2^{i-1}]$. If the highest range used is used by two or more nodes, then the algorithm fails (assuming $\beta \geq 2$). We shall bound from below the probability that exactly two nodes use the highest subrange in use; this is clearly a lower bound on the failure probability of the algorithm.

Let $X_i^v$ be the event that node $v$ transmits in the first round using the $i$-th highest subrange. Since the nodes are identical, the same probability holds for them all, so let $p_i = \Pr[X_i^v]$. Observe that the probability that no node transmits in the round is at least $1 - n\sum_i p_i$, and since that can hold with probability at most $1/n$, it follows that $\sum_i p_i \geq \frac{1}{n}(1 - \frac{1}{n})$. Let $q$ be the largest number such that

$$\sum_{i=1}^{q} p_i \leq \frac{1}{2n}. \tag{5}$$

So, a subrange of rank at least $q+1$ is in use.

Let $A_i$ be the event that at least two nodes use the $i$-th highest subrange, $B_i$ be the event that no node transmits at subranges $1, 2, \ldots, i-1$, and $C_i = A_i \cap B_i$ be the event that both $A_i$ and $B_i$ occur, for $i = 1, 2, \ldots$. Then, $C = \bigcup_i C_i$ is the event that at least two nodes use the highest subrange in use. Observe that $\Pr[A_i|B_i] \geq \Pr[A_i]$, since the non-use of the $i-1$ highest subranges only makes the event $A_i$ more likely. Then,

$$\Pr[C_i] = \Pr[A_i \cap B_i] = \Pr[A_i|B_i]\Pr[B_i] \geq \Pr[A_i]\Pr[B_i].$$

We bound the probability of $A_i$, $i \leq q$, by the first term of the binomial expansion:

$$\Pr[A_i] > \binom{n}{2} p_i^2 (1-p_i)^{n-2} > \frac{n^2}{3} p_i^2 \left(1 - \frac{1}{2n}\right)^{n-2} > \frac{n^2}{3e} p_i^2.$$

Also, applying (5),

$$\Pr[B_i] \geq 1 - n\sum_{j=1}^{i-1} p_i \geq \frac{1}{2}.$$

Observe that the $C_i$'s are mutually exclusive and apply the Cauchy-Schwarz inequality followed by (5) to obtain:

$$\Pr[C] \geq \sum_{i=1}^{q} \Pr[C_i] \geq \frac{n^2}{3e} \sum_{i=1}^{q} p_i^2 \cdot \frac{1}{2} \geq \frac{n^2}{6e} \frac{(\sum_{i=1}^{q} p_i)^2}{q} \geq \frac{1}{24e \cdot q}.$$

The algorithm fails when $C$ holds, and thus we may assume that $\Pr[C] \leq 1/n$, which implies that $q \geq n/(24e) = \Omega(n)$. Hence, the claim.

Observe that for the case of known $n$, we obtain an essentially tight bound of $2^{\tilde{\Theta}(n)}$ on the needed power range.

*Remark 2.* We note that a construction can be given in the Euclidean plane that achieves the same result but with slightly weaker power tradeoffs. It consists of $n/2$ well-separated node-pairs that are internally close. It, however, does not avail itself to easy generalizations to protocols with greater number of rounds, and is therefore omitted.

## 5 Trading Time for Power Range

In this section, we explore how much the power range can be reduced by increasing the round complexity. We present a multi-round protocol that requires limited power range and derive a lower bound on the power range required by any $t$-round leader election algorithm, for $t \geq 2$.

### 5.1 Multi-round Protocol

When a smaller power range is available, we can give a protocol that uses a larger number of rounds.

Our multi-round algorithm simply repeats the 2-round algorithm $t$ times, for a given number $t \geq 1$, but using a slower-growing power function. Namely, we change the ID-selection function to $g_t(k) = 2^k k^{3t+1}$, and the power function to $f_t(ID_v) = P \cdot ID_v^{\gamma (ID_v)^{1/t}}$. After each round-pair repetition, each competitor $v$ updates its $leader_v$ value to the *largest* among those heard so far.

First, we observe that it suffices to succeed in one of the round-pairs.

**Observation 1.** *If, in some round-pair, all receivers hear from a particular node $v$, and the senders all get informed of $v$ as a leader, then the algorithm successfully terminates with $v$ as leader.*

*Proof.* After this round-pair, all nodes have $leader_v$ value set as $w$. Thus, all broadcasts that follow use $w$ for the value of $leader_v$.

Let $U = n^{1/t}$. Suppose we can guarantee that the failure probability of an individual round-pair is at most $1/U$. Then, the probability that *all* $t$ round-pairs are unsuccessful is $1/U^t = 1/n$, as desired. Thus, it suffices to ensure that the failure probability of each round be at most $1/(2U)$. Let $Z$ be the set of competitors with the highest $k_v$ value, and recall that $|Z| \leq 8 \log n$ with probability greater than $1 - \frac{1}{8n}$, by the same argument as in Lemma 1. Observe that for success, it suffices that one node transmits with at least $n^{c\alpha}|Z| \leq n^{c\alpha+1}$ times the power of any other transmitting node, as argued in Lemma 3. A node $w$ with the highest ID will satisfy $ID_w^\gamma \geq n^{c\alpha+1}$, as $g(k_w) \geq g(k_1)$ (w.h.p.) It also holds that $ID_w^{1/t} \geq n^{1/t}$.

Thus, what remains is to argue the counterpart of Lemma 2.

**Lemma 4.** *In a given round, with probability at least $1 - 1/(2U)$, some node $w$ receives an ID such that $(ID_w)^{1/t} \geq (ID_v)^{1/t} - 1$, for all other nodes $v$.*

*Proof.* Let $Z$ be the set of competitors with the largest $k_v$-value. Recall that IDs are allocated uniformly at random, and for nodes in $Z$, the range is of size at least $g_t(k_1) = 2^{\log n - \log\log n - 2}(\log n - \log\log n - 2)^{3t+1} \geq \frac{n}{8\log n}(\frac{\log n}{2})^{3t+1} \geq \frac{1}{2^{3t+1}} n \log^{3t} n$, for large enough $n$. The probability that a given pair of nodes $u, v$ in $Z$ receive nearly equivalent IDs, with $|(ID_u)^{1/t} - (ID_v)^{1/t}| \leq 1$, is at most $g_t(k_1)^{-1/t} \leq \frac{2}{n^{1/t}\log^3 n}$. Thus, the probability that some two nodes in $Z$ receive nearly equivalent IDs is at most

$$\frac{\binom{|Z|}{2}}{g_t(k_1)^{1/t}} \leq \frac{2^3 \cdot 8^2 \log^2 n}{n^{1/t}\log^3 n} < \frac{1}{2n^{1/t}},$$

for sufficiently large $n$.

The correctness of the algorithm follows from the above observations.

**Theorem 5.** *For each number $t = O(\log n/\log\log n)$, there is a $2t$-round algorithm using a power range $2^{n^{O(1/t)}}$ that correctly elects a leader, w.h.p.*

### 5.2 Lower Bound for Multi-round Protocols

**Theorem 6.** *Any $t$-round leader election algorithm in the SINR model running correctly w.h.p. requires a power range $2^{\Omega(\sqrt[t-1]{n})}$, $t \geq 2$. This holds even if the nodes know $n$, the number of nodes in the network, and the nodes are located in a unit metric (where all distances are equal).*

*Proof.* We consider $n$ nodes located in a unit metric space. In this setting, after any round of the algorithm either all listening nodes receive a message, or no progress is made (assuming $\beta \geq 1$). Since the nodes do not operate full-duplex, any leader election algorithm requires at least two rounds, one round for the winner to broadcast its message, and one round to be informed of the victory.

Let $A$ be a $t$-round leader election algorithm in the SINR model that runs correctly with probability greater than $1 - 1/n$. Since at least two rounds of successful communication are needed, Algorithm $A$ fails when no listening node receives a message during the first $t-1$ rounds. This happens with probability $\prod_{r=1}^{t-1} p_r$, where $p_r$ denotes the probability that no listener receives a message in round $r$. Since algorithm $A$ succeeds with probability greater than $1 - 1/n$,

$$\frac{1}{n} > \prod_{r=1}^{t-1} p_r.$$

Now, consider round $r$. Let $q$ and $C$ be as in Theorem 4. We can show by a similar argument that $Pr[C] \geq \frac{1}{12e \cdot q}$, assuming $\beta \geq 2$. No listener receives a message in round $r$ when $C$ holds, and thus $\Pr[C] \leq p_r$, which implies that

$$q \geq \frac{1}{12e \cdot p_r}.$$

It follows that $1/n \geq (\frac{1}{12eq})^{t-1}$, and therefore $q \geq \sqrt[t-1]{n}/(12e) = \Omega(\sqrt[t-1]{n})$. Thus, algorithm $A$ requires a power range $2^{\Omega(\sqrt[t-1]{n})}$.

## 6 Conclusions and Acknowledgments

We have shown that power control can yield the ultimate speedup for leader election in the SINR model. This is thanks to the capture effect, which is the crucial property in which SINR differs from graphs-based models.

It would be exciting to see these techniques applied more widely. Multi-hop settings and more restricted power ranges are natural directions to examine, as well as problems beyond leader election. In general, the value of power control and the capture effect is still not fully understood.

We thank Hsin-Hao Su and Nancy Lynch for helpful comments and discussions.

## References

1. Abramson, N.: The ALOHA system: another alternative for computer communications. In: Proceedings of the Fall Joint Computer Conference, AFIPS 1970 (Fall), 17–19 November 1970, pp. 281–285. ACM, New York (1970)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in radio networks: an exponential gap between determinism randomization. In: PODC, pp. 98–108. ACM (1987)
3. Borbash, S.A., Ephremides, A.: Wireless link scheduling with power control and SINR constraints. IEEE Trans. Inf. Theory **52**(11), 5106–5111 (2006)
4. Chlamtac, I., Kutten, S.: On broadcasting in radio networks-problem analysis and protocol design. IEEE Trans. Commun. **33**(12), 1240–1246 (1985)
5. Cruz, R.L., Santhanam, A.V.: Optimal routing, link scheduling and power control in multihop wireless networks. In: INFOCOM, vol. 1, pp. 702–711 (2003)
6. Dereniowski, D., Pelc, A.: Topology recognition and leader election in colored networks. Theoret. Comput. Sci. **621**, 92–102 (2016)
7. Fineman, J.T., Gilbert, S., Kuhn, F., Newport, C.: Contention resolution on a fading channel. In: PODC, pp. 155–164. ACM (2016)
8. Gallager, R.G.: A perspective on multiaccess channels. Technical report, DTIC Document (1985)
9. Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Trans. Inf. Theor. **46**(2), 388–404 (2006)
10. Moscibroda, T., Wattenhofer, R.: The complexity of connectivity in wireless networks. In: INFOCOM (2006)
11. Newport, C.C.: Radio network lower bounds made easy. CoRR, abs/1405.7300 (2014)
12. Willard, D.E.: Log-logarithmic selection resolution protocols in a multiple access channel. SIAM J. Comput. **15**(2), 468–477 (1986)

# Token Traversal in Ad Hoc Wireless Networks via Implicit Carrier Sensing

Tomasz Jurdzinski(✉), Michal Rozanski, and Grzegorz Stachowiak

Institute of Computer Science, University of Wrocław, Wrocław, Poland
tju@cs.uni.wroc.pl

**Abstract.** Communication problems in ad hoc wireless networks have been already widely studied under the SINR model, but a vast majority of results concern networks with constraints on connectivity, so called *strongly-connected networks*. What happens if the network is not strongly-connected, e.g., it contains some long but still viable "shortcut links" connecting transmission boundaries? Even a single broadcast in such ad hoc *weakly-connected* networks with uniform transmission powers requires $\Omega(n)$ communication rounds, where $n$ is the number of nodes in the network. The best up-to-date (randomized) distributed algorithm, designed by Daum et al. [10], accomplishes broadcast task in $O(n \log^2 n)$. In this work we show a novel deterministic distributed implementation of token traversal in the SINR model with uniform transmission powers and no restriction on connectivity. We show that it is efficient even in a very harsh model of weakly-connected networks without GPS, carrier sensing and other helping features. We apply this method to span a traversal tree and accomplish broadcast in $O(n \log N)$ communication rounds, deterministically, provided nodes are equipped with unique IDs in the range $[1, N]$ for $N \geq n$. This result implies an $O(n \log n)$-round randomized solution that does not require IDs, which improves the result from [10]. The lower bound $\Omega(n \log N)$ for deterministic algorithms proved in our work shows that our result is tight without randomization. Our implementation of token traversal routine is based on a novel implicit algorithmic carrier sensing method and a new type of selectors, which might be of independent interest.

**Keywords:** Wireless ad hoc networks · SINR · Token traversal
Broadcast · Deterministic and randomized algorithms
Algorithmic carrier sensing · Selectors · BTD trees

## 1 Introduction

We study distributed algorithms in ad hoc wireless networks in the SINR model with uniform transmission powers. We consider an *ad hoc* setting, where both

The work of the first and the third author was supported by the Polish National Science Centre grant DEC-2012/07/B/ST6/01534 and the work of the second author was supported by the Polish National Science Centre grant 2014/13/N/ST6/01850.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 15–33, 2017.
https://doi.org/10.1007/978-3-319-72050-0_2

capability and knowledge of nodes are limited — nodes know only the basic parameters of the SINR model (i.e., $\alpha, \beta, \mathcal{N}$, $\mathcal{P}$, to be defined later). We assume that each node knows its distinct ID and the range of IDs $[N] = \{1, \ldots, N\}$. Such setting appears in networks without predefined infrastructure of base stations, access points, etc. It reflects various real scenarios, such as: large sets of sensors distributed in an area of rescue operation, environment monitoring, or prospective internet of things applications.

*Token traversal.* We focus on the problem of token traversal, in which a software-defined token needs to visit all (or a subset of) nodes in the network. More precisely, in the beginning there is a distinguished node, called a source, which has a status of the token owner. In each round only one node can have a status of the token owner. The ownership of the token can be passed to a neighbor via a message; in wireless network, however, it can be challenging to select an unvisited neighbor to which the token can be passed, due to ad hoc structure and interferences. The token traversal is accomplished if every participating node has been a token owner for at least one round. Token traversal is a fundamental task in distributed system, and a tool of building algorithms to solve more complex communication and computation tasks.

*Broadcast problem.* The broadcast problem was extensively studied in the model of graph-based radio networks over the years, while distributed algorithms for the SINR model have been presented only in recent years. However, all these solutions were either randomized, or relied on the assumption that nodes of a network know their own coordinates in a given metric space (GPS), or used carrier sensing capabilities or the advantage of power control (ability to change transmission power).

*Challenges and our approach.* Almost all communication algorithms analyzed in the SINR model assumed *strong connectivity* of a network. That is, connectivity of a network is guaranteed by links $(u, v)$ such that efficient transmission from $u$ to $v$ (and from $v$ to $u$) is possible provided interference at $v$ caused by other nodes of a network is limited by some fixed constant. Our aim is to provide solutions which work in the most harsh and general scenario, when connectivity might rely on weak links and thus allow for efficient transmissions only in the case of no other (or at least very small number of) transmitters in the whole network; it is called a *weakly-connectivity* model, and subsumes the strong-connectivity one. Moreover, we assume that communicating devices have very limited capabilities, in particular, they do not use randomization, availability of locations, carrier sensing, or power control. The key challenge in design of algorithms for the model considered in this paper is the assumption that nodes of a network have initially no information about network topology. The fact that nodes use a single wireless channel and therefore their messages might collide is an additional obstacle for efficient communication.

*Our results.* We present a deterministic algorithm that traverses a token along any (even weakly-connected) wireless ad hoc network, under the uniform-power

SINR, in amortized $O(\log N)$ rounds. More specifically, the token is propagated along a specific spanning tree, called a BTD (Breadth-Then-Depth) tree in time proportional to the number of participants multiplied by $O(\log N)$. It can be applied to perform broadcast in weak connectivity ad hoc networks in $O(n \log N)$ communication rounds, and is supported by a corresponding lower bound. Our result implies $O(n \log n)$ randomized algorithm with high probability (i.e., with probability polynomially close to 1), even if IDs are not available (see Sect. 7), which improves the $O(n \log^2 n)$ algorithm of Daum et al. [10].

We also introduce new tools, which might be applicable in different scenarios and problems. Firstly, inspired by Echo procedure that simulates collision detection in radio networks [33], we introduce a kind of implicit carrier sensing allowing fast testing of emptiness of sets. Secondly, in order to efficiently select nodes from dense areas of a network, we introduce a new combinatorial structure called a *witnessed strong selector*.

*Related work.* The SINR model was extensively studied recently, both from the perspective of its structural properties [17,25,26] and design of algorithms [10,13–15,18,19,21,24,27,36,37]. First wave of algorithmic research on communication under SINR constraints focused on local problems. This includes in particular the local broadcast and link scheduling [13,16,28,29,37].

Token-based algorithms were considered in related models of multiple-access channel and radio networks, e.g., [3,31]. In radio networks, an $O(\log N)$ procedure of token passing was presented in [6,34], and combined with the BTD tree traversal. In the SINR model of weak devices subsumed by and less complex than the weak-connectivity model, efficient implementation of a token was provided [31].

A few deterministic solutions are known for the broadcast problem, most of them use information about location of nodes and assume strong connectivity. Broadcast can be accomplished deterministically in time $O(D \log^2 n)$ in such setting [23,24], where $D$ is the diameter of the communication graph. The randomized results on broadcast in ad hoc settings include [10,22]. Solutions with complexity, respectively $O((D \log n) \log^{\alpha+1} g)$ and $O(D \log^2 n)$ are presented for strong connectivity networks, where $g$ is a parameter depending on the geometry of the network. Recently Halldorsson et al. [14] proposed an algorithm which can be faster assuming that nodes are equipped with some extra capabilities.

For weak connectivity networks Daum et al. have provided $\Omega(n)$ lower bound for broadcast, even in 2-broadcastable networks. They also showed that the problem can be solved in $O(n \log^2 n)$ time with high probability.

In the related multi-hop radio network model on symmetric networks, the broadcast problem is well examined [1,2,9,11,12,30,32,35].

Due to limited space, we defer some proofs to the full version of the paper.

## 2 The Network Model

We consider a wireless single-channel network consisting of nodes located on the 2-dimensional Euclidean plane, where interferences are modeled according to

SINR (*Signal-to-Interference-and-Noise Ratio*) constraints. The model is determined by fixed parameters: path loss $\alpha > 2$, threshold $\beta > 1$, ambient noise $\mathcal{N} > 0$ and transmission power $\mathcal{P}$. Given nodes $u, v$ and a set of concurrently transmitting nodes $\mathcal{T}$, the value of $SINR(v, u, \mathcal{T})$ is defined as

$$SINR(v, u, \mathcal{T}) = \frac{\mathcal{P} \cdot d(v, u)^{-\alpha}}{\mathcal{N} + \sum_{w \in \mathcal{T} \setminus \{v\}} \mathcal{P} \cdot d(w, u)^{-\alpha}} \tag{1}$$

where $d(x, y)$ denotes the distance between locations of $x$ and $y$.

A node $u$ successfully receives a message from $v$ iff $v \in \mathcal{T}$ and $SINR(v, u, \mathcal{T}) \geq \beta$, where $\mathcal{T}$ is the set of nodes transmitting at the same time. *Transmission range* is the maximal distance at which a node can be heard provided there are no other transmitters in the network. Without loss of generality we assume that the transmission range is equal to 1. This assumption implies that the relationship $\mathcal{P} = \mathcal{N}\beta$ holds. However, it does not affect generality and asymptotic complexity of presented results.

*Communication graph.* The *communication graph* $G = (V, E)$ of a given network consists of all nodes from $V$ and edges $\{v, u\}$ between nodes that are within distance of at most 1, i.e., $\{v, u\} \in E$ iff $d(u, v) \leq 1$. The communication graph, defined as above, is a *weak connectivity graph* [10,20].

*Synchronization and content of messages.* We assume that algorithms work synchronously in rounds. In a single round, a node can transmit or receive a message from other node in the network and perform local computation. A message transmitted by a node in a round might contain the original broadcast message and additional information of size $O(\log N)$.

*Knowledge of nodes.* Each node has a unique identifier from the set $[N]$, where $N > n$ and $n$ is the number of nodes in the network. The value of $n$ or its polynomial approximation is known to the nodes. Moreover, nodes know the range of IDs space $N$, and the SINR parameters – $\mathcal{P}, \alpha, \beta, \mathcal{N}$.

*Considered problems.* We consider a general *token traversal* problem defined in Sect. 1. A node $v \neq s$ starts participating in an execution of an algorithm only after receiving the first message from another node. (This is so-called *non-spontaneous wake-up* model.) We also consider the *broadcast* problem which is to deliver a message from the designated source node $s$ to all the nodes in the network, perhaps through relay nodes as not all nodes are within transmission range of the source in multi-hop networks.

*Complexity measure.* Time (or round) complexity of an algorithm is the number of rounds after which an execution of an algorithm is finished. We assume worst-case complexity measure.

*Constructive vs non-constructive solutions.* We say that an algorithm is *constructive* if the algorithm for a given value of $N$ can be built in time polynomial with respect to $N$. The algorithms delivered in this work are constructive, because of the fact that our algorithms use combinatorial structures only for the range of parameters guarantying polynomial time construction.

## 3 Preliminaries and Combinatorial Tools

The set of integers $\{1, 2, \ldots, n\}$ is denoted by $[n]$ and $\{i, i+1, \ldots j\}$ by $[i, j]$.

A *transmission schedule* is defined by a sequence $\mathcal{S} = (S_1, ..., S_t)$ of subsets of $[N]$, where the $i$th set determines nodes transmitting in the $i$th round of the schedule. That is, a node with $\mathrm{ID}(v) \in [N]$ transmits in round $i$ of an execution of $\mathcal{S}$ if and only if $v \in S_i$.

In the following, $V$ denotes the set of nodes of a network on the plane. Thus, each node $v \in V$ is determined by its identifier $\mathrm{ID}(v)$ in $[N]$ and its coordinates on the plane. In descriptions of algorithms, $\mathrm{ID}(v)$ is sometimes identified with $v$. Let $\mathcal{B}(x, r)$ denote the ball of radius $r$ around point $x$ on the plane. We identify $\mathcal{B}(x, r)$ with the set of nodes of the network that are located inside this ball on the plane. For a node $v \in V$, $N_v = \{w \in V \mid d(v, w) \leq 1\}$ denotes the set of neighbors of $v$ in the communication graph. For $a > b > 0$, $\chi(a, b)$ denotes the largest possible size of a set of points $X$ included in a ball of radius $b$ such that $d(x, y) > a$ for each distinct $x, y \in X$.

A node $w$ is in the *graph distance* $i$ from $v$ if $i$ is the length of a shortest path connecting $w$ and $v$ in the communication graph. Assume that a distinguished source node $s \in V$ is fixed. Then, $L_i \subseteq V$ denotes the set of nodes in graph distance $i$ from $s$ (layer $i$). Thus, e.g., $L_0 = \{s\}$ and $L_1 = N_s$.

We say that a node $v$ *awakes* $w$ in an execution of an algorithm if the first message successfully received by $w$ is sent by $v$.

### 3.1 Combinatorial Tools

In this section we introduce combinatorial tools applied in our token traversal algorithm. A set $S \subseteq [N]$ *selects* $x \in X$ from $X \subseteq [N]$ when $S \cap X = \{x\}$. A sequence $\mathcal{S} = (S_1, \ldots, S_t)$ of sets over $[N]$ is called $(N, k)$-strongly selective family (or $(N, k)$-ssf) if for each subset $X \subseteq [N]$ such that $|X| \leq k$, and each $x \in X$ there is $i \in [t]$ such that $S_i$ selects $x$ from $X$.

**Lemma 1** [8]. *There exists a $(N, k)$-ssf of size $O(\min\{k^2 \log(N/k), N\})$ for each $k \leq N$.*

Now, we introduce the notion of a *witnessed strong selector*, which is a generalization of strongly selective families.

*Witnessed strong selector.* A sequence $\mathcal{S} = (S_1, \ldots, S_m)$ of sets over $[N]$ satisfies *witnessed strong selection property for a set* $X \subseteq [N]$, if for each $x \in X$ and each $y \notin X$ there is a set $S_i \in \mathcal{S}$ such that $X \cap S_i = \{x\}$ and $y \in S_i$. A sequence $\mathcal{S} = (S_1, \ldots, S_m)$ is a $(N, k)$-*witnessed strong selector* (or $(N, k)$-wss) of size $m$ if for every subset $X \subseteq [N]$ of size $k$ the family $\mathcal{S}$ satisfies the witnessed strong selection property for $X$.

Note that any $(N, k)$-wss is also, by definition, an $(N, k)$-ssf. Additionally, $(N, k)$-wss guarantees that each element outside of a given set $X$ of size $k$ has to be a "witness" of selection of every element from $X$. Below we state an upper bound on the optimal size of $(N, k)$-wss.

**Lemma 2.** *For each positive integers $N$ and $k \leq N$, there exists an $(N, k)$-wss of size $O(k^3 \log N)$.*

**Construction of witnessed strong selectors.** We aim at the efficient algorithm constructing a $(N, k)$-wss for a constant $k$. Our solution is inspired by the algorithm of Clementi et al. [7], which employs the technique of conditional probabilities.

**Lemma 3.** *For each integers $0 < k < N$, a $(N, k)$-wss of size $O(k^3 \log N)$ can be constructed in time $N^{O(k)}$; in particular, it can be constructed in polynomial time for any $k = O(1)$.*

### 3.2 SINR Related Properties

We say that distinct nodes $u, v \in A$ form a *closest pair of nodes* $(u, v)$ in the set $A$ if $d(u, v) = min_{x,y \in A, x \neq y}\{d(x, y)\} \leq 1/2$.[1]

Below, we state the fact that $u$ can hear $v$ if $(u, v)$ is a closest pair, $v$ is transmitting and there is no other transmitter in distance $O(d(u, v))$, where the constant hidden in the big-O notation is determined by SINR parameters.

**Lemma 4.** *There exists a constant $\kappa_0$ (which depends merely of the SINR parameters) which satisfies the following property. Let $u, v$ be a closest pair of nodes, $d(u, v) = d < 1/2$ in $A$. If $u$ is the only transmitter in $\mathcal{B}(v, \kappa_0 \cdot d)$, then $v$ receives the message from $u$.*

Using Lemma 4, the bound on the optimal size of witnessed strong selectors (Lemma 2) and the def. of a closest pair, one can obtaind the following corollaries.

**Corollary 1.** *There exists a constant $\kappa$ (which depends merely of the SINR parameters) which satisfies the following property. Let $u, v$ be a closest pair of nodes in $A$, $d(u, v) = d < 1/2$. Then, there exists a set $A' \subseteq A$ such that $u, v \in A'$, $|A'| \leq \kappa$ and $v$ receives a message transmitted by $u$ provided no other element of $A'$ is sending a message at the same time.*

**Corollary 2.** *There exists a transmission schedule $\mathcal{S}$ of size $O(\log N)$ such that, for each closest pair $(u, v)$ from a set $A$, $u$ receives a message from $v$ during an execution of $\mathcal{S}$ on $A$.*

[1] Note that there is no closest pair in $A$ according to this definition if $d(x, y) > 1/2$ for each distinct $x, y \in A$.

## 4 High Level Idea of the Algorithm

Our token traversal algorithm builds a spanning tree of the communication graph of a network, where the source node $s$ (the initial holder of the token) is the root. Each node, after receiving the token, transmits the broadcast and awake message. If a node $u$ is awaken by $v$ (i.e., $u$ receives the first message from $v$), $u$ becomes a *child* of $v$ and $v$ is the *parent* of $u$. After sending the broadcast and awake message, the token holder learns all its newly-awaken neighbors, who have become its children. After that, the token holder passes the token sequentially to all its children. Finally, it passes the token back to its parent. The algorithm ends when the source receives the token back from all its children. A similar approach, resembling both dfs and bfs, has appeared in the context of radio networks [6] under the name Breadth-Then-Depth (BTD) search.

The most challenging part for a design of the above strategy in the model considered in this paper is to learn the children of a token holder. To this aim, we consider the full selection problem: for a given node $v$ and a set $X$ of its neighbors unknown to $v$, the node $v$ should learn the set $X$. Using appropriate novel selectors, and the idea of local leader election in the uniform SINR model [24], we can assure that full selection is done in $O(\log^2 N + |X| \log N)$ rounds. As each node becomes the child of only one other node, an application of full selection at each node (when it receives the token for the first time) would give $O\left(\sum_{v \in V}(\log^2 N + |\text{children}(v)| \log N)\right) = O(n \log^2 N)$ time algorithm. In order to improve time complexity to $O(n \log N)$, we will reduce full selection time from $O(\log^2 N + |X| \log N)$ to $O(\log N + |X| \log N)$. To this aim, we apply a kind of *implicit carrier sensing*. Thanks to that tool, we can check whether $X$ is empty in $O(\log N)$ rounds and reduce complexity of full selection to $O((|X|+1) \log N)$.

Below, we discuss the technical ingredients of our solutions in more detail.

**Network sparsification.** As discussed above, each node $v$ is supposed to determine its neighbors awaken by it after receiving the token, in the procedure called the full selection (Algorithm 5). To this aim we apply the sparsification technique presented in Sect. 5.2. Assume that $X \neq \emptyset$ is the set of neighbors of $v$ which should be learnt by $v$. The idea is to execute a short schedule (of size $O(\log N)$) on $X$ which guarantees that nodes from closest pairs (and possibly some other nodes) exchange messages. Then, a graph defined by successful exchanges of messages is built and a non-empty matching is determined in this graph. Finally, one node from each matched pair is chosen to be a member of a "sparsified" set. As such a procedure gives the sparsified set of size at most $|X|/2$, $r = O(\log |X|)$ repetitions of this procedure gives a set of size $O(1)$. Then, the only element(s) of the sparse set can report all $r$ elements matched with them. Thus, $r$ elements of $X$ are reported in amortized time $O(\log N)$ per an element. In order to implement this technique efficiently under SINR constraints, we have introduced *witnessed strong selectors* (Sect. 3.1).

**Implicit carrier sensing.** As mentioned before, we use so-called implicit carrier sensing in order to quickly verify whether the set of nodes awaken by a given node $v$ is (not) empty. More generally, implicit carrier sensing technique allows

for checking emptiness of a set $X \subseteq N_v$, provided two auxiliary nodes $v_1, v_2$ are known such that $d(v, v_1) \leq 1$, $d(v_1, v_2) \leq 1$ and $d(v, v_2) > 1$ (see Subsect. 5.1 for details). Our implementation of the token traversal algorithm will assure that $d(v, \text{parent}(v)) \leq 1$ and $d(v, \text{parent}(\text{parent}(v))) > 1$ for each $v$ in graph-distance at least 2 from the source $s$. Thus, the auxiliary nodes can be $v_1 = \text{parent}(v)$ and $v_2 = \text{parent}(\text{parent}(v))$. This however does not apply to the source $s$ (it does not have the parent) and its neighbors (there is no parent(parent($v$)) for each $v \in N_s$). Therefore, we have to handle $\{s\} \cup N_s$ separately, using less efficient emptiness test and a more complex algorithm.

## 5 Implicit Carrier Sensing and Network Sparsification

In this section we introduce key tools applied in our token-traversal algorithm: implicit carrier sensing and sparsification. We present them separately, as we think they might be applicable in other problems in wireless ad hoc networks.

### 5.1 Implicit Carrier Sensing

Consider the problem that a node $v$ is going to verify quickly whether some set $X \subseteq N_v$ is empty. Each node $x$ knows whether $x \in X$ but nodes do not have any information regarding other elements of $X$. At the end of an execution of an algorithm, $v$ should know whether $X = \emptyset$. This problem has been solved efficiently by so-called Echo procedure in the symmetric radio networks model, provided the node $v$ knows some neighbor $w \notin X$ already [33]. We develop an analogous tool for SINR networks, which provides a limited carrier sense capability.

Assume that $v, v_1, v_2$ are fixed such that $v$ is a neighbor of $v_1$ and $v_1$ is a neighbor of $v_2$. Moreover, at least one of distances $d(v, v_1)$, $d(v_1, v_2)$ is not smaller than $1/2$. Then, we can test emptiness of $X$ by checking if

- $v$ receives the message from $v_1$ when $v_1$ transmits together with $X$, and
- $v_1$ receives the message from $v_2$ when $v_2$ transmits together with $X$.

More precise description of the procedure is given as EmptinessTest below (see Algorithm 1). The first idea is that successful transmissions in both rounds correspond to the fact that $X$ is empty. However, if $X$ is small and $v$ is very close to $v_1$ or $v_1$ is very close to $v_2$, it might be the case that interference from $X$ does not prevent successful transmissions in both rounds under SINR constraints. Therefore, we need a more involved algorithm and analysis. The constant $c_{\alpha,\beta}$ in the algorithm is equal to the smallest number $c$ such that $c$ transmitting nodes located in distance (at most) 2 from a given node $u$ produce interference which prevents reception by $u$ of a message transmitted from distance $\geq 1/2$.

**Algorithm 1.** EmptinessTest$(v, v_1, v_2, X)$

Assumptions: $d(v, v_1) \leq 1$, $d(v_1, v_2) \leq 1$, $d(v, v_1) \geq 1/2$ or $d(v_1, v_2) \geq 1/2$, $X \subsetneq N_v$, $v_1, v_2 \notin X$.
Let $c_{\alpha,\beta}$ be the smallest natural number such that $\frac{P/(1/2)^\alpha}{\mathcal{N}+c_{\alpha,\beta}P/2^\alpha} < \beta$.

1: Round 1: $v_1$ and all elements of $X$ transmit a message
2: Round 2: $v_2$ and all elements of $X$ transmit a message
3: Round 3: if $v_1$ received a message in Round 2 then $v_1$ transmits a message
4: **if** $v$ received a message in Round 1 **and** $v$ received a message in Round 3 **then**
5: execute $(N, c_{\alpha,\beta})$-ssf on all elements of $X$
6: **if** $v$ received a message: return false
7: **else** return true
8: **else**
9: return false

**Lemma 5.** *EmptinessTest works in $O(\log N)$ rounds. Moreover, if $d(v, v_1) \leq 1$ and $d(v_1, v_2) \leq 1$ and ($d(v, v_1) \geq 1/2$ or $d(v_1, v_2) \geq 1/2$) then EmptinessTest$(v, v_1, v_2, X)$ returns true if and only if the set $X \subseteq N_v$ is empty.*

### 5.2 Network Sparsification

In this section we develop a tool for fast selection of elements of a set of nodes. The particular problem of *network sparsification* is as follows: given a non-empty set $X$ of nodes inside $B(v, 1)$ such that at least two nodes are within distance $1/2$, choose a subset $Y$ of $X$ such that $1 \leq |Y| \leq |X|/2$. The idea is to use a short schedule which guarantees that close neighbors exchange messages (see Corollary 2), implicitly build a graph corresponding to these two-way transmissions, choose a non-empty matching in such a graph, and select one element from each matched pair.

As a direct application of Corollary 2 does not give a satisfying time complexity, we then introduce the notion of proximity graph and show how to build it with aid of witnessed strong selectors efficiently.

*Exchange graphs.* We define the notion of *exchange graph* which describes all possible exchange of messages between nodes during an execution of a schedule $T$. For a given schedule $T$ and the set of nodes $V$, an *exchange graph* $G_T$ is a graph on $V$, such that $\{u, w\}$ is an edge in $G_T$ iff there is a successful transmission in both directions between $u$ and $w$ during $T$.

We say that a distributed protocol *builds* $G_T$ if, as a result of an execution of this protocol on a given network, each node knows its neighbors in $G_T$. Note that, after a single execution of $T$, each node $v$ knows nodes whose messages are successfully received by $v$. However, in order to determine its neighbors in $G_T$, $v$ also needs to know which nodes received its message.

In order to provide this information to all nodes, we can apply the following algorithm, called ExGraphConstruction$_T$: first, each node $v$ enumerates the senders $u_1, \ldots, u_p$ of all messages received during $T$; then, one can repeat $|T|$ times the schedule $T$, where each node transmits $u_i$ in the $i$th repetition of $T$.

**Lemma 6.** *ExGraphConstruction$_T$ builds the exchange graph $G_T$ in $O(|T|^2)$ rounds. Moreover, if the maximal degree $\delta$ of $G_T$ is known to nodes in advance, the algorithm works in $O(|T|\delta)$.*

*Proximity graphs.* The idea behind our network sparsification algorithm is to build a graph on nodes of an input set $X$ containing a closest pair as an edge, find a matching in that graph and choose one element of each matched pair as an element of the output $Y$. To do this, a fast protocol which produces a non-empty graph is needed, provided there is a closest pair in the input set of nodes. Let *proximity graph* of a given set of nodes be any graph on this set such that vertices of each closest pair $u, v$ are connected by an edge (while the graph may contain more edges).

By Corollary 1 we know that, in an execution of $(N, \kappa)$-ssf, nodes of each closest pair exchange messages. Thus, by Lemma 6, ExGraphConstruction$_{\mathbf{T}}$ builds a proximity graph in $O(\log^2 N)$ rounds, where $\mathbf{T}$ is an $(N, \kappa)$-ssf of length $O(\kappa^2 \log N) = O(\log N)$ (see Theorem 1).

Our goal is to build a proximity graph faster. Our construction builds on the following observations. First, if $u$ can hear $v$ during an execution of $T$ in a round in which $w$ is transmitting as well, then $u, w$ is for sure not a closest pair. Second, by Corollary 1, given a closest pair $(u, v)$, $u$ can hear $v$ in a round in which $v$ transmits and none of the other $\kappa$ closest to $u$ nodes transmits.

Given an $(N, \kappa)$-wss $\mathbf{S}$ for the constant $\kappa$ from Corollary 1, one can build a proximity graph of degree $\kappa = O(1)$ in $O(\log N)$ rounds using the following distributed algorithm called ProximityGraphConstruction at a node $v$:

- Execute $\mathbf{S}$.
- Determine the set $C_v$ of all nodes $u$ such that $v$ has received a message from $u$ during $\mathbf{S}$ and $v$ has not received any other message in rounds in which $u$ is transmitting (according to $\mathbf{S}$).
- If $|C_v| > \kappa$, then remove all elements from $C_v$.
- Send information about the content of $C_v$ to other nodes in consecutive $|C_v|$ repetitions of $\mathbf{S}$.
- Choose as neighbors in the final graph the set $E_v$ of all elements $w \in C_v$ st $v \in C_w$.

**Lemma 7.** *Let $X \subseteq V$ be a set of nodes. Then ProximityGraphConstruction executed on $X$ builds a proximity graph $H(X)$ of constant degree in $O(\log N)$ rounds.*

*Handshakes and sparsification.* Let $H(X)$ denote the proximity graph resulting from the ProximityGraphConstruction procedure executed by nodes from $X$. We assume that $X$ contains nodes from a closest pair, thus $H(X)$ contains at least one edge (Lemma 7). Recall that our goal in this section is to choose a nonempty subset of $X$ of size at most $|X|/2$. The idea is to build a non-empty matching on $H(X)$ and choose exactly one node per each matched pair. We say that nodes chosen by our procedure *survive*. For further applications, for each

node $v$ which survives the procedure, we store its removed counterpart in the local variable $p(v)$.

Algorithm 2 finds a non-empty matching in a proximity graph $H(X)$ build by ProximityGraphConstruction (provided the set of edges of $H(X)$ is not empty) by connecting each pair of neighbors $(v, w)$ such that $v$ is the local minimum (its ID is smaller than IDs of its neighbors) and $v$ has the smallest ID among neighbors of $w$ in $H(X)$.

**Algorithm 2.** Handshake($X$) ▷ Remark: an execution at $v \in X$

1: $v$ executes ProximityGraphConstruction($v$) using $(N, \kappa)$-wss **S** ▷ see L. 7
2: **if** $E_v = \emptyset$: $v$ does not participate in further steps.
3: $\min_v \leftarrow \min_{u \in E_v}\{\mathrm{ID}(u)\}$
4: Execute **S**, where:
if $ID(v) < \min_v$: $v$ transmits the message $m = \langle \underline{handshake}, ID(v), \min_v \rangle$
▷ $v$ is a local minimum;
if $ID(v) > \min_v$: $v$ transmits the message $m = \langle \underline{match}, ID(v), \min_v \rangle$
5: **if** $ID(v) < \min_v$ and $v$ received the message $\langle \underline{match}, \min_v, ID(v), \rangle$ **then**
6: $p(v) \leftarrow \min_v$
7: status($v$) $\leftarrow$ survived
8: **else**
9: status($v$) $\leftarrow$ eliminated
10: $v$ is switched off

**Lemma 8.** *Let $X \subseteq V$ be a subset of a network. Let $Y \subseteq X$ be the set of nodes that survived Handshake($X$) (see line 7 of Algorithm 2). If there exists a closest pair in $X$ then $1 \leq |Y| \leq |X|/2$. Moreover, for each $v \in Y$, $p(v) \in X \setminus Y$. The round complexity of Handshake procedure is $O(\log N)$.*

## 6 Token Traversal Algorithm

In this section we describe our token traversal algorithm. As it gives also immediate solution to the broadcasting problem, we present the algorithm in terms of the broadcasting task.

At the beginning of the main algorithm (Algorithm 3), the source $s$ wakes up all its neighbors (which become its children). Then, the general idea is that each node $v$, after receiving the token, learns its children (nodes awaken by $v$) using FullSelection (Algorithm 5) and passes the token to them. As our time bound for FullSelection$(v, X)$ for $v \in L_1$ and $X \subseteq N_v$ is $O(\log^2 N + |\mathrm{children}(v)| \log N)$ – see Lemma 10, this approach guarantees time $O(n \log^2 N)$ (and it might be $\Omega(n \log^2 N)$ if $|L_1| = \Omega(n)$). In order to achieve a better bound, the nodes from $L_1$ learn their children in a different way.

After the initial transmission by $s$, it learns the whole set of its neighbors $N_s = L_1$, using FullSelection. Then, $s$ allows each $v \in N_s = L_1$ to transmit separately which wakes up all elements of $L_2$ and set the parent from $L_1$ for

each element of $L_2$. The goal of HandleSecondLayer is to select all elements of $L_2$, allow each of them to transmit separately which in turn gives information to each $w \in L_1$ about all its children. As mentioned earlier, we do not want to implement this task by calling FullSelection for each $v \in L_1$, as it would increase time complexity to the order of $n \log^2 N$. We postpone the exact description of HandleSecondLayer and discuss the remaining part of the algorithm and its subroutines. After handling the second layer, a standard token traversal algorithm starts from the source (Algorithm 4), where each node from $\{s\} \cup L_1$ already knows its children, while each other node $v \notin \{s\} \cup L_1$ learns children($v$) using FullSelection.

Algorithm 3 contains pseudo-code of our main algorithm. Then, procedures called in the main algorithm are presented in the top-down fashion.

*Remark.* In pseudo-codes, we use set theoretic operations, e.g., $A \leftarrow X \setminus Y$. Such notation describes local decisions of nodes and means that each $x$ knows whether it belongs to $X$ and $Y$ and therefore it can determine if it belongs to $A$.

**Algorithm 3.** BroadcastWithToken($s$)

Initially for each node $u$ parent($u$) $= \perp$ and layer($v$) $= 0$.
1: Transmit $\langle \underline{\text{hello}}, s \rangle$
2: FullSelection($s, L_1$)
3: HandleSecondLayer
4: TokenTraversal($s$)

**if** a node $w$ receives $\langle \underline{\text{hello}}, v \rangle$ and parent($w$) $= \perp$ **then**
    parent($w$) $\leftarrow v$
    layer($w$) $\leftarrow$ layer($v$) $+ 1$

**Algorithm 4.** TokenTraversal($v$)

1: Transmit $\langle \underline{\text{hello}}, v \rangle$
2: **if** layer($v$) $> 1$ **then**
3:  FullSelection($v, \{w \mid \text{parent}(w) = v\}$)
4: **for each** $w \in$ children($v$) **do**
5:  Transmit $\langle \underline{\text{token}}, w \rangle$ ▷ pass the token to $w$
6:  Wait until receiving a message $\langle \underline{\text{release}}, v \rangle$ ▷ Token is back at $v$
7: Transmit $\langle \underline{\text{release}}, \text{parent}(v) \rangle$ ▷ pass the token to the parent of $v$

In the following, we describe the main subroutine FullSelection called at each node $v \notin L_1$. FullSelection($v, X$) repeats procedure PartialSelection several times, until all elements of $X$ are selected, i.e., each of them transmits a message received by $v$. An execution of PartialSelection($v, X$) results in reporting $r > 0$

elements of $X$ in $O(r \log N)$ rounds, provided $X$ is not empty. In the case that $X$ is empty, PartialSelection$(v, X)$ ends in $O(\log N)$ rounds when $v \notin \{s\} \cup L_1$ and in $O((\log n) \cdot (\log N))$ rounds otherwise.

The procedure PartialSelection (Algorithm 6) executes Handshake several times. (An alternative for partial selection might be e.g. by $(N, k, k/2)$-selectors [4] for $k = 2, 4, 8, \ldots, 2^{\log n}$. However, no constructive solutions with optimal size are known for them.) Handshake$(X)$ sparsifies the set $X$. As Handshake is always executed on $X \subseteq N_v$ for a reference node $v$, $X$ is contained in a ball of radius 1. Let $m = |X|$. If $m$ is smaller than $\chi(1/2, 2)$, then each element of $X$ transmits separately (see line 6 of Algorithm 6). Otherwise, there exists a closest pair of nodes in $X$, and some elements are removed from $X$ such that the size of $X$ after the execution is in the range $[1, m/2]$ (see Lemma 8). In this way at least one element of $X$ is selected in $O(\log |X|)$ executions of Handshake.

However, our goal is to select one element per each execution of Handshake on the average. Fortunately, each node $v$ which survives the $i$th execution of Handshake has associated the unique element $p(v)$ which has survived the first $i-1$ executions of Handshake and has not survived the $i$th execution (see Lemma 8). The node $v$ stores such elements in $P(v)$.

**Algorithm 5.** FullSelection$(v, X)$ ▷ $v$ learns all elements of $X$

1: $Y \leftarrow X$, $w \leftarrow v$, children$(v) \leftarrow \emptyset$
2: **while** $w \neq \perp$ **do**
3: $w \leftarrow$ PartialSelection$(v, X)$
4: $Y \leftarrow P(w)$ ▷ if $w \neq \perp$, $w$ broadcasts $P(w)$ in $|P(w)| + 1$ rounds
5: $X \leftarrow X \setminus Y$
6: children$(v) \leftarrow$ children$(v) \cup Y$ ▷ $v$ learns $Y$ in step 4

**Lemma 9.** *1. Assume that $X \subseteq N_v$ is not empty. Then, PartialSelection$(v, X)$ is finished after $O(r \log N)$ rounds for $0 < r \leq \log n$ and $v$ has received a message from $w \in X$ such that $P(w) \subseteq X$ and $|P(w)| = r$.*
*2. PartialSelection$(v, X)$ for $X = \emptyset$ works in $O(\log N)$ rounds for $v \notin L_0 \cup L_1$ and in $O((\log n) \cdot (\log N))$ rounds for $v \in L_0 \cup L_1$. Moreover, $v$ is aware of the fact that $X = \emptyset$ after the execution of PartialSelection$(v, X)$ for $X = \emptyset$.*

To summarize, we state the following properties of FullSelection.

**Lemma 10.** *Let $X \subseteq N_v$. Then, $v$ knows all elements of $X$ after an execution of FullSelection$(v, X)$. Moreover, each $u \in X$ transmits uniquely at some round of FullSelection$(v, X)$. The execution time is $O(|X| \log N + f(N))$, where $f(N) = O(\log N)$ if $v \notin L_0 \cup L_1$ and $f(N) = O((\log n) \cdot (\log N))$ otherwise.*

**Handling the second layer.** Recall that each node from $L_0 \cup L_1$ is the only transmitter in some round during steps 1. and 2. of the main algorithm (Algorithm 3). The nodes from $L_2$ are awaken in this way and they know their parents from $L_1$.

**Algorithm 6.** PartialSelection$(v, X)$ ▷ Assumption: $X \subseteq N_v$

1: **if** $v \notin L_0 \cup L_1$ **then** ▷ $L_0 = \{s\}$, $L_1 = N_s$
2: **if** EmptinessTest$(v, \text{parent}(v), \text{parent}(\text{parent}(v)), X)$: return $\perp$
3: $r \leftarrow 1$
4: **for each** $w \in X$**:** $P(w) \leftarrow \emptyset$
5: **repeat**
6: Execute $(N, k)$-ssf on $X$ for $k = \lceil \chi(1/2, 1) \rceil + 1$.
7: **if** $v$ received a message from some node $w$ during step 6 **then**
8: return $w$ ▷ i.e., $v$ transmits a message which ends the procedure
9: Handshake$(X)$
10: **for each** $w$**: if** $w$ survived Handshake$(X)$: $P(w) \leftarrow P(w) \cup \{p(w)\}$
11: **for each** $w$**: if** $w$ did not survive Handshake$(X)$: $w$ remove itself from $X$
12: $r \leftarrow r + 1$
13: **until** $r = \log N$ ▷ until $r = \log n$ if $n$ is known
14: return $\perp$

Now, we describe HandleSecondLayer subroutine which assures that each node $v \in L_2$ is a unique transmitter in some round (and it transmits ID of its parent in each transmission). In this way the nodes from $L_1$ learn about their children.

To achieve the above stated goal, we repeat the following procedure. First, the leader $v$ in $L_2$ is elected and, all elements of $N_v \cap L_2$ are selected using FullSelection$(v, N_v \cap L_2)$. Then, all selected elements are removed from consideration and the process is repeated until no unselected elements in $L_2$ remain. As the consecutive leaders are in distance $> 1$ to each other, this process finishes after at most $\chi(1, 2)$ elections of the leader. More formal presentation of this idea is given in Algorithm 7.

**Algorithm 7.** HandleSecondLayer ▷ Assumption: each $v$ knows if $v \in L_2$

1: $L \leftarrow L_2$ ▷ Initially, $L$ is the second layer
2: $c \leftarrow \chi(1, 2)$
3: **for** i=1,2,. . . ,c **do**
4: $v \leftarrow$ Leader$(L)$
5: $v$ transmits $\langle \underline{\text{leader}}, v \rangle$
6: $X \leftarrow \{w \mid w$ received the message $\underline{\text{leader}}\}$
7: FullSelection$(v, X)$
8: $L \leftarrow L \setminus X$

Now, we provide an efficient implementation of leader election in line 4 of Algorithm 7. We require that exactly one element $x \in X$ has the status *leader* as a result of a *leader election* algorithm. (Observe that we do not require that all elements of $X$ know ID of the node with status *leader*.)

The idea of the leader election procedure (Algorithm 8) is to repeat Handshake several times in order to sparsify the input set $X$, as in PartialSelection. In this way, some node $w \in X$ will be the only transmitter at some round, after $O(\log N)$ repetitions of Handshake. The problem is that the unique transmitter $w$ might be unaware of its uniqueness. If all elements of $X$ were also in $N_v$ for a distinguished node $v$, then $v$ could confirm reception of a message from $w$ and inform in this way $w$ that it was the leader[2]. As the set of nodes $X \subset L_2$ executing the leader election procedure is not necessarily a subset of $N_v$ for any $v$, PartialSelection does not give a solution to the leader election problem directly. Instead, we use the fact that each node $v \in X$ knows parent$(v) \in L_1$. We assure that each node $v \in L_1$ that hears its child $w \in L_2$, tries to report this fact to the source. If the source node receives such a message at some time, it chooses $w$ to be the leader and passes this information through $v = \text{parent}(w)$ to $w$. This is guaranteed to happen when $w$ was the unique transmitter in $L_2$, and it will happen eventually for some $w \in L_2$ in at most $\log n$) executions of the for-loop in Algorithm 8.

**Algorithm 8.** Leader$(L)$ ▷ Assumption: parent$(x) \in L_1$, $x \in L_2$ for each $x \in L$

1: leader$(v) \leftarrow$ false for all $v \in L$
2: elected $\leftarrow$ false ▷ elected is a local variable stored at $s$
3: **for** $i = 1, 2, \ldots, \log n$ **do**
4: Execute $(N, k)$-ssf for $k = \lceil \chi(1/2, 2) \rceil + 1$, each transmission is followed by:
**Round 1:**
**if** $w \in L_1$ received a message from its child $x \in L_2$:
$w$ transmits $\langle \underline{\text{leader-proposal}}, x, w \rangle$
**Round 2:**
**if** elected=false and $s$ received $\langle \underline{\text{leader-proposal}}, x, y \rangle$:
$s$ transmits $\langle \underline{\text{leader-elect}}, x, y \rangle$; elected $\leftarrow$ true
**Round 3:**
**if** $w \in L_1$ received a message $\langle \underline{\text{leader-elect}}, x, w \rangle$: $w$ transmits $\langle \underline{\text{leader-elect}}, x \rangle$.
5: **if** $x \in L$ received a message $\langle \underline{\text{leader-elect}}, x \rangle$ in Round 3: leader$(x) \leftarrow$ true
6: Handshake$(L)$
7: $L \leftarrow$ nodes from $L$ which survived Handshake$(L)$

**Proposition 1.** *Let $L \subseteq L_2$ be a set of nodes such that parent(parent$(x)$) $= s$ and parent$(x) \in L_1 \cap N_x$ for each $x \in L$. Then, Leader$(L)$ solves the leader election problem on $L$ in $O((\log N) \cdot (\log n))$ rounds.*

Given the leader election procedure, we can prove that each node from $L_2$ is the only transmitter in some round of HandleSecondLayer. This in turn gives information to nodes from $L_1$ about their children.

[2] Note that, under SINR constraints, $v$ can receive a message from some node $x$ even when $x$ is not the unique transmitter in a round. However, as $v$ "selects" the leader and announces its choice, this does not cause any problem with uniqueness of the leader.

**Lemma 11.** *Assume that parent$(v) = s$ for each $v \in L_1$ and parent$(u) \in L_1 \cap N_u$ for each $u \in L_2$. Then, each $u \in L_2$ is the only transmitter in some round of an execution of HandleSecondLayer. Moreover, HandleSecondLayer works in in $O(n \log N)$ rounds. otherwise.*

**Theorem 1.** *BroadcastWithToken solves weak connectivity broadcast in the ad hoc SINR model in $O(n \log N)$ rounds.*

*Proof.* Lemmas 10 and 11 imply that steps 1–3 of the algorithm are finished in time stated in the theorem. Then, the TokenTraversal algorithm builds a spanning tree of a network, the token is passed once over each edge of this tree in each direction which altogether takes $O(n)$ rounds. Moreover, for each node $v \notin L_0 \cup L_1$, FullSelection$(v, X)$ is executed when $v$ receives the token for the first time, for $X$ equal to the set of children of $v$. An execution of FullSelection$(v, X)$ takes $O((|X|+1) \log N)$ rounds, where $X$ is the set of selected elements. As each $w \in V \setminus (L_0 \cup L_1)$ is only once an element of $X$ in an execution of FullSelection$(v, X)$ (when $v =$ parent$(w)$), the overall complexity of all executions of FullSelection during TokenTraversal$(s)$ is $O(n \log N)$.

## 7 Lower Bound and Extensions

**Lower bound.** Employing a similar approach as in [5], we build a network of linear diameter such that it takes at least $\Omega(n \log N)$ rounds to broadcast a message.

**Theorem 2.** *For any deterministic algorithm $\mathcal{A}$ and $n < N/6$, there exists a network $\mathbf{N}$ of size $3n+1$ on the plane such that it takes $\Omega(n \log N)$ rounds before $\mathcal{A}$ completes the broadcast in $\mathbf{N}$ in a weakly connected SINR network.*

**Randomized algorithm.** In [10] the authors proposed a randomized algorithm that solves broadcast in time $O(n \log^2 n)$ and a lower bound of $\Omega(n)$. Our result fits into the scenario presented therein provided each node picks a random ID in range $[1, n^3]$ and performs the deterministic algorithm, which works as long as the IDs are different (this is true with high probability). Thus, our solution is $O(n \log n)$.

**Constructive solution.** We need $(N, k)$-wss only for constant values of parameter $k$. Thus, by Lemma 3, the actual algorithm for fixed $N$ can be determined in time polynomial with respect to $N$.

## 8 Conclusions

We presented a novel token traversal algorihm in weakly-connected ad hoc networks under the uniform-power SINR model, leading to asymptotically optimal deterministic spanning tree and broadcast algorithm and a nearly-optimal randomized solution improving [10]. The token traversal occurred to be efficient for spanning tree and broadcast problem, therefore we conjecture that it could play a substantial role in algorithmics in general class of ad hoc wireless networks.

**Acknowledgments.** The authors would like to thank Darek Kowalski for fruitful discussions and his comments on the paper.

## References

1. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A lower bound for radio broadcast. J. Comput. Syst. Sci. **43**(2), 290–298 (1991)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: an exponential gap between determinism and randomization. J. Comput. Syst. Sci. **45**(1), 104–126 (1992)
3. Bienkowski, M., Jurdzinski, T., Korzeniowski, M., Kowalski, D.R.: Distributed online and stochastic queuing on a multiple access channel. In: Aguilera, M.K. (ed.) DISC 2012. LNCS, vol. 7611, pp. 121–135. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33651-5_9
4. Bonis, A.D., Gasieniec, L., Vaccaro, U.: Optimal two-stage algorithms for group testing problems. SIAM J. Comput. **34**(5), 1253–1270 (2005)
5. Bruschi, D., Pinto, M.D.: Lower bounds for the broadcast problem in mobile radio networks. Distrib. Comput. **10**(3), 129–135 (1997)
6. Chlebus, B.S., Kowalski, D.R., Pelc, A., Rokicki, M.A.: Efficient distributed communication in ad-hoc radio networks. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 613–624. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_49
7. Clementi, A.E.F., Crescenzi, P., Monti, A., Penna, P., Silvestri, R.: On computing ad-hoc selective families. In: Goemans, M., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX/RANDOM-2001. LNCS, vol. 2129, pp. 211–222. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44666-4_24
8. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: SODA 2001, pp. 709–718 (2001)
9. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. In: FOCS, pp. 492–501. IEEE Computer Society (2003)
10. Daum, S., Gilbert, S., Kuhn, F., Newport, C.: Broadcast in the ad hoc SINR model. In: Afek, Y. (ed.) DISC 2013. LNCS, vol. 8205, pp. 358–372. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41527-2_25
11. De Marco, G.: Distributed broadcast in unknown radio networks. SIAM J. Comput. **39**(6), 2162–2175 (2010)
12. Emek, Y., Gasieniec, L., Kantor, E., Pelc, A., Peleg, D., Su, C.: Broadcasting in UDG radio networks with unknown topology. Distr. Comput. **21**(5), 331–351 (2009)
13. Goussevskaia, O., Moscibroda, T., Wattenhofer, R.: Local broadcasting in the physical interference model. In: DIALM-POMC 2008, pp. 35–44. ACM (2008)
14. Halldórsson, M.M., Holzer, S., Lynch, N.A.: A local broadcast layer for the SINR network model. In: PODC 2015, ADM, pp. 129–138 (2015)
15. Halldórsson, M.M., Mitra, P.: Nearly optimal bounds for distributed wireless scheduling in the SINR model. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 625–636. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_50
16. Halldórsson, M.M., Mitra, P.: Towards tight bounds for local broadcasting. In: Kuhn, F., Newport, C.C. (eds.) FOMC, p. 2. ACM (2012)
17. Halldórsson, M.M., Tonoyan, T.: How well can graphs represent wireless interference? In: STOC 2015, pp. 635–644 (ACM)

18. Hobbs, N., Wang, Y., Hua, Q.-S., Yu, D., Lau, F.C.M.: Deterministic distributed data aggregation under the SINR model. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 385–399. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29952-0_38
19. Jurdzinski, T., Kowalski, D.R.: Distributed backbone structure for algorithms in the SINR model of wireless networks. In: Aguilera, M.K. (ed.) DISC 2012. LNCS, vol. 7611, pp. 106–120. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33651-5_8
20. Jurdzinski, T., Kowalski, D.R.: Distributed randomized broadcasting in wireless networks under the SINR model. In: Kao, M.-Y. (ed.) Encyclopedia of Algorithms, pp. 577–580. Springer, New York (2016). https://doi.org/10.1007/978-1-4939-2864-4_604
21. Jurdzinski, T., Kowalski, D.R., Rozanski, M., Stachowiak, G.: Distributed randomized broadcasting in wireless networks under the SINR model. In: Afek, Y. (ed.) DISC 2013. LNCS, vol. 8205, pp. 373–387. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41527-2_26
22. Jurdzinski, T., Kowalski, D.R., Rozanski, M., Stachowiak, G.: On the impact of geometry on ad hoc communication in wireless networks. In: PODC 2014, pp. 357–366. ACM (2014)
23. Jurdzinski, T., Kowalski, D.R., Stachowiak, G.: Distributed deterministic broadcasting in uniform-power ad hoc wireless networks. In: Gąsieniec, L., Wolter, F. (eds.) FCT 2013. LNCS, vol. 8070, pp. 195–209. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40164-0_20
24. Jurdzinski, T., Kowalski, D.R., Stachowiak, G.: Distributed deterministic broadcasting in wireless networks of weak devices. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7966, pp. 632–644. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_55
25. Kantor, E., Lotker, Z., Parter, M., Peleg, D.: The minimum principle of SINR: a useful discretization tool for wireless communication. In: FOCS 2015, pp. 330–349. IEEE (2015)
26. Kantor, E., Lotker, Z., Parter, M., Peleg, D.: The topology of wireless communication. J. ACM **62**(5), 37:1–37:32 (2015)
27. Kesselheim, T.: A constant-factor approximation for wireless capacity maximization with power control in the SINR model. In: SODA 2011, pp. 1549–1559. SIAM (2011)
28. Kesselheim, T.: Dynamic packet scheduling in wireless networks. In: PODC 2012, pp. 281–290. ACM (2012)
29. Kesselheim, T., Vöcking, B.: Distributed contention resolution in wireless networks. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 163–178. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15763-9_16
30. Kowalski, D.R.: On selection problem in radio networks. In: PODC 2005, pp. 158–166. ACM (2005)
31. Kowalski, D.R., Moses Jr., W.K., Vaya, S.: Deterministic backbone creation in an SINR network without knowledge of location. CoRR, abs/1702.02460 (2017)
32. Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad hoc radio networks. In: PODC 2003, pp. 73–82. ACM (2003)
33. Kowalski, D.R., Pelc, A.: Faster deterministic broadcasting in ad hoc radio networks. SIAM J. Discrete Math. **18**(2), 332–346 (2004)
34. Kowalski, D.R., Pelc, A.: Time of deterministic broadcasting in radio networks with local knowledge. SIAM J. Comput. **33**(4), 870–891 (2004)

35. Kushilevitz, E., Mansour, Y.: An $\Omega(DLog(N/D))$ lower bound for broadcast in radio networks. In: PODC 1993, pp. 65–74. ACM (1993)
36. Moscibroda, T., Wattenhofer, R.: The complexity of connectivity in wireless networks. In: INFOCOM 2006. IEEE (2006)
37. Yu, D., Hua, Q., Wang, Y., Lau, F.C.M.: An o(log n) distributed approximation algorithm for local broadcasting in unstructured wireless networks. In: DCOSS 2012, pp. 132–139. IEEE (2012)

# Identifiers and Labelling

# Short Labeling Schemes for Topology Recognition in Wireless Tree Networks

Barun Gorain[1(✉)] and Andrzej Pelc[2]

[1] Indian Institute of Information Technology Vadodara, Gandhinagar 382028, Gujarat, India
baruniitg123@gmail.com

[2] Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada
pelc@uqo.ca

**Abstract.** We consider the problem of topology recognition in wireless (radio) networks modeled as undirected graphs. Topology recognition is a fundamental task in which every node of the network has to output a map of the underlying graph i.e., an isomorphic copy of it, and situate itself in this map. In wireless networks, nodes communicate in synchronous rounds. In each round a node can either transmit a message to all its neighbors, or stay silent and listen. At the receiving end, a node $v$ hears a message from a neighbor $w$ in a given round, if $v$ listens in this round, and if $w$ is its only neighbor that transmits in this round. Nodes have labels which are (not necessarily different) binary strings. The length of a labeling scheme is the largest length of a label. We concentrate on wireless networks modeled by trees, and we investigate two problems.

- What is the shortest labeling scheme that permits topology recognition in all wireless tree networks of diameter $D$ and maximum degree $\Delta$?
- What is the fastest topology recognition algorithm working for all wireless tree networks of diameter $D$ and maximum degree $\Delta$, using such a short labeling scheme?

We are interested in deterministic topology recognition algorithms. For the first problem, we show that the minimum length of a labeling scheme allowing topology recognition in all trees of maximum degree $\Delta \geq 3$ is $\Theta(\log\log\Delta)$. For such short schemes, used by an algorithm working for the class of trees of diameter $D \geq 4$ and maximum degree $\Delta \geq 3$, we show almost matching bounds on the time of topology recognition: an upper bound $O(D\Delta)$, and a lower bound $\Omega(D\Delta^{\epsilon})$, for any constant $\epsilon < 1$.

Our upper bounds are proven by constructing a topology recognition algorithm using a labeling scheme of length $O(\log\log\Delta)$ and using time $O(D\Delta)$. Our lower bounds are proven by constructing a class of trees for which any topology recognition algorithm must use a labeling scheme of length at least $\Omega(\log\log\Delta)$, and a class of trees for which any topology recognition algorithm using a labeling scheme of length $O(\log\log\Delta)$ must use time at least $\Omega(D\Delta^{\epsilon})$, on some tree of this class.

A. Pelc—Partially supported by NSERC discovery grant 8136–2013 and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 37–52, 2017.
https://doi.org/10.1007/978-3-319-72050-0_3

**Keywords:** Topology recognition · Wireless network
Labeling scheme · Feasibility · Tree · Time

## 1 Introduction

**The model and the problem.** Learning the topology of an unknown network by its nodes is a fundamental distributed task in networks. Every node of the network has to output a map of the underlying graph, i.e., an isomorphic copy of it, and situate itself in this map. Topology recognition can be considered as a preprocessing procedure to many other distributed algorithms which require the knowledge of important parameters of the network, such as its size, diameter or maximum degree. It can also help to determine the feasibility of some tasks that depend, e.g., on symmetries existing in the network.

We consider wireless networks, also known as radio networks. Such a network is modeled as a simple undirected connected graph $G = (V, E)$. As it is usually assumed in the algorithmic theory of radio networks [2,12,13], all nodes start simultaneously and communicate in synchronous rounds. In each round, a node can either transmit a message to all its neighbors, or stay silent and listen. At the receiving end, a node $v$ hears a message from a neighbor $w$ in a given round, if $v$ listens in this round, and if $w$ is its only neighbor that transmits in this round. We do not assume collision detection: if more than one neighbor of a node $v$ transmits in a given round, node $v$ does not hear anything (except the background noise that it also hears when no neighbor transmits).

In this paper, we restrict attention to wireless networks modeled by trees, and we are interested in deterministic topology recognition algorithms. Topology recognition is formally defined as follows. Every node $v$ of a tree $T$ must output a tree $T'$ and a node $v'$ in this tree, such that there exists an isomorphism $f$ from $T$ to $T'$, for which $f(v) = v'$. Topology recognition is impossible, if nodes do not have any a priori assigned labels, because then any deterministic algorithm forces all nodes to transmit in the same rounds, and no communication is possible. Hence we consider labeled networks. A *labeling scheme* for a network represented by a tree $T = (V, E)$ is any function $\mathcal{L}$ from the set $V$ of nodes into the set $S$ of finite binary strings. The string $\mathcal{L}(v)$ is called the label of the node $v$. Note that labels assigned by a labeling scheme are not necessarily distinct. The *length* of a labeling scheme $\mathcal{L}$ is the maximum length of any label assigned by it. We investigate two problems.

- What is the shortest labeling scheme that permits topology recognition in all wireless tree networks of diameter $D$ and maximum degree $\Delta$?
- What is the fastest topology recognition algorithm working for all wireless tree networks of diameter $D$ and maximum degree $\Delta$, using such a short labeling scheme?

**Our results.** For the first problem, we show that the minimum length of a labeling scheme allowing topology recognition in all trees of maximum degree $\Delta \geq 3$ is $\Theta(\log \log \Delta)$. For such short schemes, used by an algorithm working

for the class of trees of diameter $D \geq 4$ and maximum degree $\Delta \geq 3$, we show almost matching bounds on the time of topology recognition: an upper bound $O(D\Delta)$, and a lower bound $\Omega(D\Delta^{\epsilon})$, for any constant $\epsilon < 1$.

Our upper bounds are proven by constructing a topology recognition algorithm using a labeling scheme of length $O(\log\log\Delta)$ and using time $O(D\Delta)$. Our lower bounds are proven by constructing a class of trees for which any topology recognition algorithm must use a labeling scheme of length at least $\Omega(\log\log\Delta)$, and a class of trees for which any topology recognition algorithm using a labeling scheme of length $O(\log\log\Delta)$ must use time at least $\Omega(D\Delta^{\epsilon})$, on some tree of this class.

These main results are complemented by establishing complete answers to both problems for very small values of $D$ or $\Delta$. For trees of diameter $D = 3$ and maximum degree $\Delta \geq 3$, the fastest topology recognition algorithm using a shortest possible scheme (of length $\Theta(\log\log\Delta)$) works in time $\Theta(\frac{\log\Delta}{\log\log\Delta})$. The same holds for trees of diameter $D = 2$ and maximum degree *at most* $\Delta$, for $\Delta \geq 3$. Finally, if $\Delta = 2$, i.e., for the class of lines, the shortest labeling scheme permitting topology recognition is of constant length, and the best time of topology recognition using such a scheme for lines of diameter (length) at most $D$ is $\Theta(\log D)$.

Our results should be contrasted with those from [11], where topology recognition was studied in a different model. The authors of [11] considered wired networks in which there are port numbers at each node, and communication proceeds according to the $\mathcal{LOCAL}$ model [21], where in each round neighbors can exchange all available information without collisions. In this model, they showed a simple topology recognition algorithm working for a labeling scheme of length 1 in time $O(D)$. Thus there was no issue of optimality: both the length of the labeling scheme and the topology recognition time for such a scheme were trivially optimal. Hence the authors focused on tradeoffs between the length of (longer) schemes and the time of topology recognition. In our scenario of wireless networks, the labeling schemes must be longer and algorithms for such schemes must be slower, in order to overcome collisions.

**Related work.** Algorithmic problems in radio networks modeled as graphs were studied for such tasks as broadcasting [2,13], gossiping [2,12] and leader election [19]. In some cases [2,12] the topology of the network was unknown, in others [13] nodes were assumed to have a labeled map of the network and could situate themselves in it.

Providing nodes of a network or mobile agents circulating in it with information of arbitrary type (in the form of binary strings) that can be used to perform network tasks more efficiently has been proposed in [1,3–10,14,16–18,20]. This approach was referred to as algorithms using *informative labeling schemes*, or equivalently, algorithms with *advice*. When advice is given to nodes, two variations are considered: either the binary string given to nodes is the same for all of them [15] or different strings may be given to different nodes [9,11], as in the case of the present paper. If strings may be different, they can be considered as labels assigned to nodes. Several authors studied the minimum size of advice (length

of labels) required to solve the respective network problem in an efficient way. The framework of advice or labeling schemes permits to quantify the amount of information that nodes need for an efficient solution of a given network problem, regardless of the type of information that is provided.

In [3] the authors investigated the minimum size of advice that has to be given to nodes to permit graph exploration by a robot. In [18], given a distributed representation of a solution for a problem, the authors investigated the number of bits of communication needed to verify the legality of the represented solution. In [7] the authors compared the minimum size of advice required to solve two information dissemination problems, using a linear number of messages. In [8] the authors established the size of advice needed to break competitive ratio 2 of an exploration algorithm in trees. In [9] it was shown that advice of constant size permits to carry on the distributed construction of a minimum spanning tree in logarithmic time. In [12] short labeling schemes were constructed with the aim to answer queries about the distance between any pair of nodes. In [5] the advice paradigm was used for online problems. In the case of [20] the issue was not efficiency but feasibility: it was shown that $\Theta(n \log n)$ is the minimum size of advice required to perform monotone connected graph clearing. In [16] the authors studied radio networks for which it is possible to perform centralized broadcasting in constant time. This is the only paper studying the size of advice in the context of radio networks. In [11] the authors studied the task of topology recognition in wired networks with port numbers. The differences between this scenario and our setting of radio networks, in the context of topology recognition, was discussed in the previous section.

## 2 Preliminaries and Organization

Throughout the paper, $D$ denotes the diameter of the tree and $\Delta$ denotes its maximum degree. The problem of topology recognition is non-trivial only for $D, \Delta \geq 2$, hence we make this assumption from now on.

According to the definition of labeling schemes, a label of any node should be a finite binary string. For ease of comprehension, we present our labels in a more structured way, as either finite sequences of binary strings, or pairs of such sequences, where each of the component binary strings is later used in the topology recognition algorithm in a particular way. It is well known that a sequence $(s_1, \ldots, s_k)$ of binary strings or a pair $(\sigma_1, \sigma_2)$ of such sequences can be unambiguously coded as a single binary string whose length is a constant multiple of the sum of lengths of all binary strings $s_i$ that compose it. Hence, presenting labels in this more structured way and skipping the details of the encoding does not change the order of magnitude of the length of the constructed labeling schemes.

Let $T$ be any rooted tree with root $r$, and let $L(T)$ be a labeling scheme for this tree. We say that a node $u$ in $T$ *reaches* $r$ within time $\tau$ using algorithm $\mathcal{A}$ if there exists a simple path $u = u_0, u_1, \cdots, u_{k-1}, u_k = r$ and a sequence of integers $t_0 < t_1 < \cdots < t_{k-1} \leq \tau$, such that in round $t_i$, the node $u_i$ is the only

child of its parent $u_{i+1}$ that transmits and the node $u_{i+1}$ does not transmit in round $t_i$, according to algorithm $\mathcal{A}$.

We define the history $H(\mathcal{A}, \tau)$ of the root $r$ of the tree $T$ as the labeled subtree of $T$ which is spanned by all the nodes that reach $r$ within time $\tau$, using algorithm $\mathcal{A}$. The history $H(\mathcal{A}, \tau)$ is the total information that node $r$ can learn about the tree $T$ in time $\tau$, using algorithm $\mathcal{A}$.

## 3 A Lower Bound on the Length of Labeling Schemes

As mentioned in the Introduction, topology recognition without any labels cannot be performed in any tree because no information can be successfully transmitted in an unlabeled radio network. Hence, the length of a labeling scheme permitting topology recognition must be a positive integer. In this section we show a lower bound $\Omega(\log\log\Delta)$ on the length of labeling schemes that permit topology recognition for all trees with maximum degree $\Delta \geq 3$.

Let $S$ be a star with the central node $r$ of degree $\Delta$. Denote one of the leaves of $S$ by $a$. For $\lfloor \frac{\Delta}{2} \rfloor \leq i \leq \Delta - 1$, we construct a tree $T_i$ by attaching $i$ leaves to $a$. The maximum degree of each tree $T_i$ is $\Delta$. Let $\mathcal{T}$ be the set of trees $T_i$, for $\lfloor \frac{\Delta}{2} \rfloor \leq i \leq \Delta - 1$. Hence the size of $\mathcal{T}$ is at least $\frac{\Delta}{2}$.

The following result shows that any labeling scheme allowing topology recognition in trees of maximum degree $\Delta$ must have length $\Omega(\log\log\Delta)$.

**Theorem 1.** *For any tree $T \in \mathcal{T}$ consider a labeling scheme LABEL(T). Let TOPO be any topology recognition algorithm that solves topology recognition for every tree $T \in \mathcal{T}$ using the scheme LABEL(T). Then there exists a tree $T' \in \mathcal{T}$, for which the length of the scheme LABEL(T') is $\Omega(\log\log\Delta)$.*

## 4 Time for Maximum Degree $\Delta \geq 3$ and Diameter $D \geq 4$

In this section, we present our main results on the time of topology recognition, using the shortest possible labeling schemes (those of length $\Theta(\log\log\Delta)$) for trees of maximum degree $\Delta \geq 3$ and diameter $D \geq 4$. We propose an algorithm using a labeling scheme of length $\Theta(\log\log\Delta)$ and working in time $O(D\Delta)$, and prove an almost matching lower bound $\Omega(D\Delta^{\epsilon})$ on the time of such schemes, for any constant $\epsilon < 1$.

### 4.1 The Main Algorithm

Let $T$ be a rooted tree of diameter $D$ and maximum degree $\Delta$. It has either a central node or a central edge, depending on whether $D$ is even or odd. If $D$ is even, then the central node is the unique node in the middle of every simple path of length $D$, and if $D$ is odd, then the central edge is the unique edge in the middle of every simple path of length $D$. For the sake of description, we choose the central node or one of the endpoints of the central edge as the root $r$ of $T$.

Let $h = \lceil D/2 \rceil$ be the height of this tree. The *level* of any node $v$ is its distance from the root. For any node $v$ we denote by $T_v$ the subtree of $T$ rooted at $v$.

We propose an algorithm that solves topology recognition in time $O(D\Delta)$, using a labeling scheme of length $O(\log\log\Delta)$. The structure of the tree will be transmitted bottom up, so that the root learns the topology of the tree, and then transmits it to all other nodes. The main difficulty is to let every node know the round number $\rho$ in which it has to transmit, so that it is the only node among its siblings that transmits in round $\rho$, and consequently its parent gets the message. Due to very short labels, $\rho$ cannot be explicitly given to the node as a part of its label. We overcome this difficulty by carefully coding $\rho$ for a node $v$, using the labels given to the nodes of the subtree rooted at $v$, so that $v$ can unambiguously decode $\rho$.

A node $v$ in $T$ is called *heavy*, if $|V(T_v)| \geq \frac{1}{4}(\lfloor\log\Delta\rfloor + 1)$. Otherwise, the node is called *light*. Note that the root is a heavy node. For a heavy node $v$, choose a subtree $T'_v$ of $T_v$ rooted at $v$, of size $\lceil\frac{1}{4}(\lfloor\log\Delta\rfloor + 1)\rceil$.

First, we define the labeling scheme $\Lambda$. The label $\Lambda(v)$ of each node $v$ contains two parts. The first part is a vector of markers that are binary strings of constant length, used to identify nodes with different properties. The second part is a vector of 5 binary strings of length $O(\log\log\Delta)$ that are used to determine the time when the node should transmit.

Below we describe how the markers are assigned to different nodes of $T$.

1. Mark the root $r$ by the marker 0, and mark one of the leaves at maximum depth by the marker 1.
2. Mark all the nodes in $T'_r$ by the marker 2.
3. Mark every heavy node by the marker 3, and mark every light node by the marker 4.
4. For every heavy node $v$ all of whose children are light, mark all the nodes of $T'_v$ by the marker 5.
5. For every light node $v$ whose parent is heavy, mark all the nodes in $T_v$ by the marker 6.

The first part of every label is a binary string $M$ of length 7, where the markers are stored. Note that a node can be marked by multiple markers. If the node is marked by the marker $i$, for $i = 0, \ldots, 6$, we have $M(i) = 1$; otherwise, $M(i) = 0$.

In order to describe the second part of each label, we define an integer $t_v$ for every heavy node $v \neq r$, and an integer $z_v$, for every light node $v$ whose parent is heavy. We define $t_v$, for a heavy node $v$ at level $l > 0$, to identify the time slot in which $v$ will transmit according to the algorithm. The definition is by induction on $l$. For $l = 1$, let $v_1, v_2, \ldots, v_x$, be the heavy children of $r$. Set $t_{v_i} = i$. Suppose that $t_v$ is defined for every heavy node $v$ at level $l$. Let $v$ be a heavy node at level $l$. Let $u_1, u_2, \ldots, u_y$ be the heavy children of $v$. We set $t_{u_1} = t_v$, and we define $t_{u_j}$, for $2 \leq j \leq y$, as distinct integers from the range $\{1, \ldots, y\} \setminus \{t_v\}$. This completes the definition of $t_v$, for all heavy nodes $v \neq r$.

We now define $z_v$, for a light node $v$ whose parent is heavy, to identify the time slot in which $v$ will transmit according to the algorithm. Let $S_i$ be

a maximal sequence of non-isomorphic rooted trees of $i$ nodes. There are at most $2^{2(i-1)}$ such trees. Let $\mathcal{S}$ be the sequence which is the concatenation of $S_1, S_2, \ldots, S_{\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil - 1}$. Let $q$ be the length of $\mathcal{S}$. Then $q \leq 2^{2(\frac{1}{4}(\lfloor \log \Delta \rfloor + 1))} \leq \sqrt{2\Delta}$. Note that the position of any tree of $i$ nodes in $\mathcal{S}$ is at most $2^{2i-1}$. Let $\mathcal{S} = (T_1, T_2, \ldots, T_q)$. For a light node $v$ whose parent is heavy, we define $z_v = k$, if $T_v$ and $T_k$ are isomorphic.

The second part of each label is a vector $L$ of length 5, whose terms $L(i)$ are binary strings of length $O(\log \log \Delta)$. Initialize all terms $L(i)$ for every node $v$ to 0. We now describe how some of these terms are changed for some nodes. They are defined as follows.

1. All the nodes which get $M(2) = 1$ are the nodes of $T'_r$. There are exactly $\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil$ nodes in $T'_r$. All nodes in $T'_r$ are assigned distinct ids which are binary representations of the integers 1 to $\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil$. Let $s$ be the string of length $(\lfloor \log \Delta \rfloor + 1)$ which is the binary representation of the integer $\Delta$. Let $b_1, b_2, \cdots, b_{\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil}$ be the substrings of $s$, each of length at most 4, such that $s$ is the concatenation of the substrings $b_1, b_2, \cdots, b_{\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil}$. The term $L(0)$ corresponding to a node whose id is $i$, is set to the pair $(B(i), b_i)$, where $B(i)$ is the binary representation of the integer $i$. The intuitive role of the term $L(0)$ is to code the integer $\Delta$ in the nodes of the tree $T'_r$.
2. Let $v$ be a node with $M(3) = 1$, and $M(5) = 1$, i.e., let $v$ be a heavy node whose all children are light. All nodes in $T'_v$ are assigned distinct ids which are binary representations of integers 1 to $\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil$. Let $s$ be the string of length $(\lfloor \log \Delta \rfloor + 1)$ which is the binary representation of the integer $t_v$. Let $b_1, b_2, \cdots, b_{\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil}$ be the substrings of $s$, each of length at most 4, such that $s$ is the concatenation of the substrings $b_1, b_2, \cdots, b_{\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil}$. The term $L(1)$ corresponding to a node whose id is $i$, is set to the pair $(B(i), b_i)$, where $B(i)$ is the binary representation of the integer $i$. The intuitive role of the term $L(1)$ is to code the integer $t_v$, for a heavy node $v$ whose all children are light, in the nodes of the tree $T'_v$.
3. Let $v$ be a node with $M(3) = 1$, i.e., a heavy node. Let $u$ be the parent of $v$. If $t_u = t_v$, set $L(2) = 1$ for the node $v$. The intuitive role of the term $L(2)$ at a heavy node $v$ is to tell its parent $u$ what is the value of $t_u$.
4. Let $v$ be a node with $M(4) = 1$ and $M(6) = 1$, i.e., let $v$ be a light node whose parent is heavy. All nodes in $T_v$ are assigned distinct ids which are binary representations of the integers 1 to $p$, where $p$ is the size of $T_v$. Let $s$ be the string of length at most $2p$ which is the binary representation of the integer $z_v$. Let $b_1, b_2 \cdots, b_p$ be the substrings of $s$, each of length at most 2, such that $s$ is the concatenation of the substrings $b_1$, $b_2$ $\cdots$, $b_p$. The term $L(3)$ of the node whose id is $i$ is set to the pair $(B(i), b_i)$, where $B(i)$ is the binary representation of the integer $i$. The intuitive role of the term $L(3)$ is to code the integer $z_v$, for a light node $v$ whose parent is heavy, in the nodes of the tree $T_v$.
5. Let $v$ be a node with $M(3) = 1$, i.e., a heavy node. Partition all light children $u$ of $v$ into sets with the same value of $z_u$. Consider any set $\{u_1, u_2, \ldots, u_a\}$ in this partition. Let $s$ be the binary representation of the integer $a$ and let

$b_1, b_2, \cdots, b_{\lceil \frac{1}{4}(\lfloor \log a \rfloor + 1)\rceil}$ be the substrings of $s$, each of length at most 4, such that $s$ is the concatenation of the substrings $b_1, b_2, \cdots, b_{\lceil \frac{1}{4}(\lfloor \log a \rfloor + 1)\rceil}$.

For node $u_i$, where $i \leq \lceil \frac{1}{4}(\lfloor \log a \rfloor + 1)\rceil$, the term $L(4)$ is set to the pair $(B(i), b_i)$, where $B(i)$ is the binary representation of the integer $i$, for $1 \leq i \leq \lfloor \log a \rfloor + 1$, and $b_i$ is the $i$th bit of the binary representation of $a$. The intuitive role of the term $L(4)$ is to force two light children $v_1$ and $v_2$ of the same heavy parent, such that $z_{v_1} = z_{v_2}$, to transmit in different rounds.

6. For any node $v$ the term $L(5)$ is set to the binary representation of the integer $\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil$. This term will be used in a gossiping algorithm that plays the role of a subroutine in our algorithm.

Notice that the length of each $L(j)$ defined above is of length $O(\log \log \Delta)$ for every node, and there is no ambiguity in setting these terms, as every term for a node is modified at most once. This completes the description of our labeling scheme whose length is $O(\log \log \Delta)$.

**Algorithm** `Tree Topology Recognition`
The algorithm consists of four procedures, namely Procedure `Parameter Learning`, Procedure `Slot Learning`, Procedure `T-R` and Procedure `Final`. They are called in this order by the algorithm. In the first two procedures we will use the simple gossiping algorithm `Round-Robin` which enables nodes of any graph of size at most $m$ with distinct ids from the set $\{1, \ldots, m\}$ to gossip in time $m^2$, assuming that they know $m$ and that each node with id $i$ has an initial message $\mu_i$. The time segment $1, \ldots, m^2$ is partitioned into $m$ segments of length $m$, and the node with id $i$ transmits in the $i$th round of each segment. In the first time segment, each node with id $i$ transmits the message $(i, \mu_i)$. In the remaining $m - 1$ time segments, nodes transmit all the previously acquired information. Thus at the end of algorithm `Round-Robin`, all nodes know the entire topology of the network, with nodes labeled by pairs $(i, \mu_i)$.

**Procedure** `Parameter Learning`
The aim of this procedure is for every node of the tree to learn the maximum degree $\Delta$, the level of the tree to which the node belongs, and the height $h$ of the tree.

The procedure consists of two stages. The first stage is executed in rounds $1, \ldots, m^2$, where $m = \lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil$, and consists of performing algorithm `Round-Robin` by the nodes with $M(2) = 1$, i.e., the nodes in $T'_r$. Each such node uses its id $i$ written in the first component of the term $L(0)$, uses its label as $\mu_i$, and takes $m$ as the integer whose representation is given in the term $L(5)$.

After this stage, the node with $M(0) = 1$, i.e., the root $r$, learns all pairs $(B(1), b_1), \ldots, (B(m), b_m)$, where $B(i)$ is the binary representation of the integer $i$, corresponding to the term $L(0)$ at the respective nodes. It computes the concatenation $s$ of the strings $b_1, b_2, \ldots, b_m$. This is the binary representation of $\Delta$.

The second stage of the procedure starts in round $m^2 + 1$. In round $m^2 + 1$, the root $r$ transmits the message $\mu$ that contains the value of $\Delta$. A node $v$, which receives the message $\mu$ at time $m^2 + i$ for the first time, sets its level as $i$ and

transmits $\mu$. When the node $u$ with $M(1) = 1$, i.e., a deepest leaf, receives $\mu$ in round $m^2 + j$, it sets its level as $h = j$, learns that the height of the tree is $h$, and transmits the pair $(h, h)$ in the next round. Every node at level $l$, after receiving the message $(h, l + 1)$ (from a node of level $l + 1$) learns $h$ and transmits the pair $(h, l)$. After receiving the message $(h, 1)$, the root $r$ transmits the message $\mu'$ that contains the value $h$. Every node learns $h$ after receiving it for the first time and retransmits $\mu'$, if its level is less than $h$. The stage, and hence the entire procedure, ends in round $m^2 + 3h$.

**Procedure** `Slot Learning`
The aim of this procedure is for every heavy node all of whose children are light, and for every light node whose parent is heavy, to learn the time slot in which it should transmit. Moreover, at the end of the procedure, every light node $v$ learns $T_v$.

Let $t_0 = m^2 + 3h$, where $m = \lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1) \rceil$. The total number of rounds reserved for this procedure is $2m^2$. The procedure starts in round $t_0 + 1$ and ends in round $t_0 + 2m^2$. The procedure consists of two stages. The first stage is executed in rounds $t_0 + 1, \ldots, t_0 + m^2$, and consists of performing algorithm `Round-Robin` by the nodes with $L(1) \neq 0$, i.e., the nodes in $T'_v$, for a heavy node $v$ all of whose children are light. Each such node uses its id $i$ written in the first component of the term $L(1)$, uses its label as $\mu_i$, and takes $m$ as the integer whose representation is given in the term $L(5)$. After this stage, each node $v$ with $M(3) = 1$ and $M(5) = 1$, i.e., a heavy node all of whose children are light, learns all pairs $(B(1), b_1), \ldots, (B(m), b_m)$, where $B(i)$ is the binary representation of the integer $i$, corresponding to the term $L(1)$ at the respective nodes. It computes the concatenation $s$ of the strings $b_1$, $b_2$, ..., $b_m$. This is the binary representation of the integer $t_v$, which will be used to compute the time slot in which node $v$ will transmit in the next procedure.

The second stage is executed in rounds $t_0 + m^2 + 1, \ldots, t_0 + 2m^2$, and consists of performing algorithm `Round-Robin` by the nodes with $L(2) \neq 0$, i.e., the nodes in $T_v$, for a light node $v$ whose parent is heavy. Each such node uses its id $i$ written in the first component of the term $L(3)$, uses its label as $\mu_i$, and takes $m$ as the integer whose representation is given in the term $L(5)$. After this stage, each node $v$ with $M(4) = 1$ and $M(6) = 1$, i.e., a light node whose parent is heavy, learns all pairs $(B(1), b_1), \ldots, (B(k), b_k)$, where $k < m$ and $B(i)$ is the binary representation of the integer $i$, corresponding to the term $L(3)$ at the respective nodes. Node $v$ computes the concatenation $s$ of the strings $b_1$, $b_2$, ..., $b_k$. This is the binary representation of the integer $z_v$, which will be used to compute the time slot in which node $v$ will transmit in the next procedure. Moreover, each node $w$ in $T_v$ learns $T_w$ because it knows the entire tree $T_v$ with all id's. The stage, and hence the entire procedure, ends in round $t_1 = t_0 + 2m^2$.

**Procedure** `T-R`
The aim of this procedure is learning the topology of the tree by the root.

All heavy nodes and all light nodes whose parent is heavy transmit in this procedure. The procedure is executed in $h$ epochs. The number of rounds reserved for an epoch is $2\Delta$. The first $\Delta$ rounds of an epoch are reserved for transmissions

of heavy nodes and the last $\Delta$ rounds of an epoch are reserved for transmissions of light nodes whose parent is heavy. The epoch $j$ starts in round $t_1+2(j-1)\Delta+1$ and ends in round $t_1 + 2j\Delta$. All the nodes at level $h - i + 1$ which are either heavy nodes or light nodes with a heavy parent transmit in the epoch $i$. When a node $v$ transmits in some epoch, it transmits a message $(\Lambda(v), T_v, C)$, where $C = t_v$, if $v$ is a heavy node, and $C = 0$, if it is a light node. Below we describe the steps that a node performs in the execution of the procedure, depending on its label.

Let $v$ be a node with $M(4) = 1$ and $M(6) = 1$, i.e., $v$ is a light node whose parent is heavy. The node $v$ transmit in this procedure if $L(4) \neq 0$. Let the level of $v$ (learned in the execution of Procedure `Parameter Learning`) be $l$. Let the first component of the term $L(4)$ be the binary representation of the integer $c > 0$. The node $v$ already knows the value $z_v$ which it learned in the execution of Procedure `Slot Learning`. Knowing $\Delta$, node $v$ computes the list $\mathcal{S} = (T_1, T_2, \ldots, T_q)$ of trees (defined above) which unambiguously depends on $\Delta$. The node $v$ transmits the message $(\Lambda(v), T_{z_v}, 0)$ in round $t_1 + 2(h - l)\Delta + \Delta + (z_v - 1)\lceil \frac{1}{4}(\lfloor \log \Delta \rfloor + 1)\rceil + c$. We will show that node $v$ is the only node among its siblings that transmits in this round.

Let $v$ be a node with $M(3) = 1$ and $M(5) = 1$, i.e., $v$ is a heavy node all of whose children are light. Let $l$ be the level of $v$. All the children of $v$ are light nodes with a heavy parent. They are at level $l - 1$. Let $u_1, u_2, \ldots, u_k$ be those children from which $v$ received messages in the previous epoch. First, the node $v$ partitions the nodes $u_1$, $u_2$, ..., $u_k$ into disjoint sets $R_1, R_2, \cdots, R_e$ such that all nodes in the same set have sent the message with same tree $Q$. For each such set $R_d$, $1 \leq d \leq e$, let $Q_d$ be the tree sent by nodes from $R_d$. The node $v$ got all pairs $(B(1), b_1), \ldots, (B(x), b_x)$, where $x = |R_d| < m$ and $B(i)$ is the binary representation of the integer $i$, corresponding to the term $L(4)$ at its children in $R_d$. Node $v$ computes the concatenation $s$ of the strings $b_1, b_2, \ldots, b_k$. Let $y_d$ be the integer whose binary representation is $s$. After computing all $y_d$'s, for $1 \leq d \leq e$, $v$ computes the tree $T_v$, by attaching $y_d$ copies of the tree $Q_d$ to $v$ for $d = 1, \ldots, e$. The node $v$ transmits the message $(\Lambda(v), T_v, t_v)$ in round $t_1 + 2(h - l)\Delta + t_v$. We will show that node $v$ is the only node among its siblings that transmits in this round.

Let $v$ be a node with $M(3) = 1$ and $M(5) = 0$, i.e., $v$ is a heavy node who has at least one heavy child. Let $u_1, \ldots, u_{k_1}$ be the light children of $v$ from which $v$ received a message in the previous epoch, and let $u'_1, \ldots, u'_{k_2}$ be the heavy children of $v$ from which $v$ received a message in the previous epoch. The node $v$ computes the tree $T_v$ rooted at $v$ as follows. It first attaches trees rooted at its light children, using the messages it received from them, in the same way as explained in the previous case. Then, it attaches trees rooted at its heavy children. These trees are computed from the code $\beta$ in the message from each of the heavy children of $v$. Let $u'$ be the unique heavy child of $v$ for which the term $L(5) = 1$. The node $v$ computes $t_v$ which is equal to the term $C$ in the message it received from the node $u'$. The node $v$ transmits the message $(\Lambda(v), T_v, t_v)$ in round $t_1 + 2(h - l)\Delta + t_v$. We will show that node $v$ is the only node among its siblings that transmits in this round.

**Procedure** `Final`
The aim of this procedure is for every node of the tree to learn the topology of the tree and to place itself in the tree. The procedure starts in round $t_1+2h\Delta+1$ and ends in round $t_1+2h\Delta+h$. In round $t_1+2h\Delta+1$, the root $r$ transmits the message that contains the tree $T_r$. In general, every node $v$ transmits a message exactly once in Procedure `Final`. This message contains the sequence $(T_r, T_{w_p}, \ldots, T_{w_1}, T_v)$, where $w_i$ is the ancestor of $v$ at distance $i$. In view of the fact that every node $v$ already knows $T_v$ at this point, after receiving a message containing the sequence $(T_r, T_{w_p}, \ldots, T_{w_1})$ in round $j$, a node $v$ transmits the sequence $(T_r, T_{w_p}, \ldots, T_{w_1}, T_v)$ in round $j+1$, if its level is less than $h$.

A node $v$ outputs the tree $T_r$, and identifies itself as one of the nodes in $T_r$ for which the subtrees rooted at their ancestors in each level starting from the root are isomorphic to the trees in the sequence $(T_r, T_{w_p}, \ldots, T_{w_1}, T_v)$. (Notice that there may be many such nodes). The procedure ends in round $t_1+2h\Delta+h$, when all nodes place themselves in $T_r$ and output $T_r$.

**Theorem 2.** *Upon completion of Algorithm* `Tree Topology Recognition`*, all nodes of a tree correctly output the topology of the tree and place themselves in it. The algorithm uses labels of length $O(\log\log\Delta)$ and works in time $O(D\Delta)$, for trees of maximum degree $\Delta$ and diameter $D$.*

### 4.2 The Lower Bound

In this section, we prove that any topology recognition algorithm using a labeling scheme of length $O(\log\log\Delta)$ must use time at least $\Omega(D\Delta^\epsilon)$, for any constant $\epsilon < 1$, on some tree of diameter $D \geq 4$ and maximum degree $\Delta \geq 3$. We split the proof of this lower bound into three parts, corresponding to different ranges of the above parameters, as the proof is different in each case.

**Case 1: $\Delta$ bounded, $D$ unbounded.** In this case we need to show a lower bound $\Omega(D)$.

**Lemma 1.** *Let $D \geq 4$ be any integer, let $\Delta \geq 3$ be any integer constant and let $c > 1$ be any real constant. For any tree $T$ of maximum degree $\Delta$ consider a labeling scheme LABEL(T) of length at most $c\log\log\Delta$. Let TOPO be any algorithm that solves topology recognition for every tree $T$ of maximum degree $\Delta$ using the labeling scheme LABEL(T). Then there exists a tree $T$ of maximum degree $\Delta$ and diameter $D$ for which TOPO must take time $\Omega(D)$.*

**Case 2: $\Delta$ unbounded, $D$ bounded.** In this case, we need to show a lower bound $\Omega(\Delta^\epsilon)$, for any constant $\epsilon < 1$. The following lemma proves a stronger result.

**Lemma 2.** *Let $\Delta \geq 3$ be any integer, let $D \geq 4$ be any integer constant, and let $c > 0$ be any real constant. For any tree $T$ of maximum degree $\Delta$, consider a labeling scheme LABEL(T) of length at most $c\log\log\Delta$. Let TOPO be an algorithm that solves topology recognition for every tree of maximum degree $\Delta$*

*and diameter $D$ using the labeling scheme LABEL(T). Then there exists a tree $T$ of maximum degree $\Delta$ and diameter $D$ for which TOPO must take time $\Omega(\frac{\Delta}{(\log \Delta)^c})$.*

**Case 3: unbounded $\Delta$ and $D$.** Let $\Delta \geq 3$, $D \geq 4$ be integers. We first assume that $D$ is even. The case when $D$ is odd will be explained later. It is enough to prove the lower bound for $D \geq 6$. Let $h = \lfloor \frac{D}{6} \rfloor$ and $g = \frac{D}{2} - h$. Then $2h \leq g \leq 2h + 2$. Let $P$ be a line of length $g$ with nodes $v_1, v_2, \cdots, v_{g+1}$, where $v_1$ and $v_{g+1}$ are the endpoints of $P$. We construct from $P$ a class of trees called *sticks* as follows.

Let $x = (x_1, x_2, \cdots, x_g)$ be a sequence of integers, with $0 \leq x_i \leq \Delta - 2$. Construct a tree $P_x$ by attaching $x_i$ leaves to the node $v_i$ for $1 \leq i \leq g$. Let $\mathcal{P}$ be the set of all sticks constructed from $P$. Then $|\mathcal{P}| = (\Delta - 1)^g$. Let $\mathcal{P} = \{P_1, P_2, \cdots, P_{(\Delta-1)^g}\}$.

Let $S$ be a rooted tree of height $h$, with root $r$ of degree $\Delta - 1$, and with all other non-leaf nodes of degree $\Delta$. The nodes in $S$ are called *basic nodes*. Let $Z = \{w_1, w_2, \cdots, w_z\}$, where $z = (\Delta - 1)^h$, be the set of leaves of $S$. Consider a sequence $y = (y_1, y_2, \cdots, y_z)$, for $1 \leq y_i \leq (\Delta-1)^g$. We construct a tree $T_y$ from $S$ by attaching to it the sticks in the following way: each leaf $w_i$ is identified with the node $v_1$ of the stick $P_{y_i}$, for $1 \leq i \leq z$. We will say that the stick $P_{y_i}$ is *glued* to node $w_i$. The diameter of each tree $T_y$ is $D$. For odd $D$, do the above construction for $D - 1$ and attach one additional node of degree 1 to one of the leaves.

Let $\mathcal{T}(\Delta, D)$ be a maximal set of pairwise non-isomorphic trees among the trees $T_y$. Then, $|\mathcal{T}(\Delta, D)| \geq \frac{((\Delta-1)^g)^z}{z!} \geq \frac{((\Delta-1)^g)^z}{z!} \geq (\Delta - 1)^{h(\Delta-1)^h}$.

Consider any time $\tau > 0$. For any tree $T \in \mathcal{T}(\Delta, D)$, consider any labeling scheme $L(T)$ and let $\mathcal{A}$ be any algorithm that solves topology recognition in every tree $T \in \mathcal{T}(\Delta, D)$ in time $\tau$, using the labeling scheme $L(T)$. The following lemma gives an upper bound on the number of basic nodes that can belong to a history of the root $r$.

**Lemma 3.** *Let $B$ be the number of basic nodes of level $i$ that can reach $r$ within time $\tau$, according to algorithm $\mathcal{A}$. Then $B \leq \frac{\tau^i}{i!}$ if $\tau \geq i$, and $B = 0$, otherwise.*

The next lemma gives a lower bound on the time of topology recognition for the class $\mathcal{T}(\Delta, D)$.

**Lemma 4.** *Let $\epsilon < 1$ be any positive real constant, and let $c > 1$ be any real constant. For any tree $T \in \mathcal{T}(\Delta, D)$, consider a labeling scheme LABEL(T) of length at most $c \log\log \Delta$. Then there exist integers $\Delta_0, D_0 > 0$ such that any algorithm that solves topology recognition for every tree $T \in \mathcal{T}(\Delta, D)$, where $\Delta \geq \Delta_0$ and $D \geq D_0$, using the scheme LABEL(T), must take time $\Omega(D\Delta^\epsilon)$ for some tree $T \in \mathcal{T}(\Delta, D)$.*

*Proof.* Wee first do the proof for even $D$. Consider an algorithm *TOPO* that solves topology recognition for every tree $T \in \mathcal{T}(\Delta, D)$ in time $\tau \leq (\frac{D}{6} - 1)\Delta^\epsilon \leq h\Delta^\epsilon$ with a labeling scheme $LABEL(T)$ of length at most $c \log\log \Delta$. For a

scheme of this length, there are at most $2^{c \log\log \Delta+1} = 2(\log \Delta)^c$ different possible labels. According to Lemma 3, for $1 \leq i \leq h$ the number of basic nodes of level $i$, that reach $r$ within time $\tau$ is at most $\frac{\tau^i}{i!}$, if $\tau \geq i$, otherwise there are no such nodes.

Denote by $q$ the total number of basic nodes that reach $r$ within time $\tau$. If $\tau \geq h$, then $q \leq \sum_{i=1}^{h} \frac{\tau^i}{i!} \leq h\frac{\tau^h}{h} = h\frac{(h\Delta^\epsilon)^h}{h!}$. We know that $\log(h!) = h\log h - \frac{h}{\ln 2} + \frac{1}{2}\log h + O(1) \geq h\log h - \frac{h}{\ln 2}$. Since $\ln 2 > \frac{1}{2}$, we have $\log(h!) > h\log h - 2h$. Therefore, $h! > \frac{h^h}{2^{-2h}}$, and hence $q \leq h\Delta^{h\epsilon}2^{2h}$. If $\tau < h$, then $q \leq \sum_{i=1}^{\tau} \frac{\tau^i}{i!} \leq \tau\Delta^{\tau\epsilon}2^{2\tau} \leq h\Delta^{h\epsilon}2^{2h}$. Therefore, $q \leq h\Delta^{h\epsilon}2^{2h}$, for all $\tau > 0$.

The number of different unlabeled sticks is at most $(\Delta-1)^{2h+2}$. Nodes of each such stick can be labeled with labels of length at most $\lfloor c\log\log\Delta \rfloor$ in at most $(2(\log\Delta)^c)^{(2h+2)\Delta}$ ways, because each stick can have at most $(2h+2)\Delta$ nodes. Therefore, the number of different labeled sticks is at most $p = (\Delta-1)^{2h+2}\ (2(\log\Delta)^c)^{(2h+2)\Delta}$.

The history of the root $r$ of a tree $T \in \mathcal{T}(\Delta, D)$ may include some nodes from a stick in $T$ only if the basic node at level $h$ to which this stick is glued is a node in the history. The maximum information that the root can get from a basic node $v$ at level $h$, but not from any other node at this level, is the information about the whole labeled stick glued to $v$.

The number of possible histories $H(TOPO, \tau)$ of the node $r$ is at most the product of the number of possible labelings of the basic nodes in $H(TOPO, \tau)$ and the number of possible gluings of labeled sticks to them. Since there are at most $q$ basic nodes in $H(TOPO, \tau)$, there are at most $(2(\log\Delta)^c)^q$ possible labelings of these nodes. Since there are at most $p$ labeled sticks to choose from, the number of possible gluings of labeled sticks to the basic nodes in $H(TOPO, \tau)$ is at most $p^q$. Therefore, the number of possible histories $H(TOPO, \tau)$ of the node $r$ is at most $2^q(\log\Delta)^{cq}p^q = (2p(\log\Delta)^c)^q$. Let $X = (2p(\log\Delta)^c)^q$. We have $\log X = q(\log p + 1 + c\log\log\Delta) = q + q\log p + qc\log\log\Delta$. Also, $\log p = (2h+2)\log(\Delta-1) + (2h+2)\Delta(1 + c\log\log\Delta)$. Therefore, $\log X = q(1 + \log p + c\log\log\Delta) = q\,(1 + (2h+2)\log(\Delta-1) + (2h+2)\Delta(1 + c\log\log\Delta) + c\log\log\Delta) \leq 5qc(2h+2)\Delta\log\Delta \leq 5h\Delta^{h\epsilon+1}2^{2h}c(2h+2)\log\Delta$. Also, $\log|\mathcal{T}(\Delta, D)| \geq h(\Delta-1)^h\log(\Delta-1)$. Now, for any $\Delta$ and for sufficiently large $h$, we have $5h\Delta^{h\epsilon+1}2^{2h}c(2h+2) < \frac{1}{2}h\Delta^h$. Therefore, $5h\Delta^{h\epsilon+1}2^{2h}c(2h+2)\log\Delta < \frac{1}{2}h\Delta^h\log\Delta < h(\Delta-1)^h\log(\Delta-1)$, for sufficiently large $\Delta$ and sufficiently large $h$.

It follows that, for sufficiently large $h$ and $\Delta$, we have $\log X < \log|\mathcal{T}(\Delta, D)|$. Therefore, there exist integers $\Delta_0$ and $D_0$ such that $X < |\mathcal{T}(\Delta, D)|$, for all $\Delta \geq \Delta_0$ and $D \geq D_0$. Hence, for $\Delta \geq \Delta_0$ and $D \geq D_0$, there exist two trees $T_1$ and $T_2$ in $\mathcal{T}(\Delta, D)$ whose roots have the same history. Therefore, the root $r$ in $T_1$ and the root $r$ in $T_2$ output the same tree as the topology, within time $\tau$. This is a contradiction, which proves the lemma for even $D$. For odd $D$, the same proof works with $D$ replaced by $D-1$. □

Lemmas 1, 2, and 4 imply the following theorem.

**Theorem 3.** *Let $\epsilon < 1$ be any positive real number. For any tree $T$ of maximum degree $\Delta \geq 3$ and diameter $D \geq 4$, consider a labeling scheme of length $O(\log\log\Delta)$. Then any topology recognition algorithm using such a scheme for every tree $T$ must take time $\Omega(D\Delta^{\epsilon})$ for some tree.*

## 5 Time for Small Maximum Degree $\Delta$ or Small Diameter $D$

In this section we solve our problem for the remaining cases of small parameters $\Delta$ and $D$, namely, in the case when $\Delta \leq 2$ or $D \leq 3$. We start with the case of small diameter $D$.

### 5.1 Diameter $D = 3$

**Theorem 4.** *The optimal time for topology recognition in the class of trees of diameter $D = 3$ and maximum degree $\Delta \geq 3$, using a labeling scheme of length $\Theta(\log\log\Delta)$, is $\Theta(\frac{\log\Delta}{(\log\log\Delta)})$.*

### 5.2 Diameter $D = 2$

We now consider the case of trees of diameter 2, i.e., the class of stars. Since there is exactly one star of a given maximum degree $\Delta$, the problem of topology recognition for $D = 2$ and a given maximum degree $\Delta$ is trivial. A meaningful variation of the problem for $D = 2$ is to consider all trees (stars) of maximum degree *at most* $\Delta$, for a given $\Delta$.

**Theorem 5.** *The optimal time for topology recognition in the class of trees of diameter $D = 2$ (i.e., stars) and maximum degree at most $\Delta$, where $\Delta \geq 3$, using a labeling scheme of length $\Theta(\log\log\Delta)$, is $\Theta(\frac{\log\Delta}{(\log\log\Delta)})$.*

### 5.3 Maximum Degree $\Delta = 2$

We finally address the case of trees of maximum degree $\Delta = 2$, i.e., the class of lines. Since there is exactly one line of a given diameter $D$, the problem of topology recognition for $\Delta = 2$ and for a given diameter $D$ is trivial. A meaningful variation of the problem for $\Delta = 2$ is to consider all trees (lines) of diameter *at most* $D$, for a given $D$.

We first propose a topology recognition algorithm for all lines of diameter at most $D$, where $D \geq 4$, using a labeling scheme of length $O(1)$ and working in time $O(\log D)$.

**Algorithm** `Line-Topology-Recognition`
Let $T$ be a tree of maximum degree 2 and diameter at most $D$, i.e., a line of length at most $D$. Let $v_1, v_2, \ldots, v_{k+1}$, for $k \leq D$, be the nodes of $T$, where $v_1$ and $v_{k+1}$ are the two endpoints. At a high level, we partition the line into

segments of length $O(\log k)$ and assign labels, containing (among other terms) couples of bits, to the nodes in each segment. This is done in such a way that the concatenation of the first bits of the couples in a segment is the binary representation of the integer $k$, and the concatenation of the second bits of the couples in a segment is the binary representation of the segment number. In time $O(\log k)$, every node learns the labels in each segment, and computes $k$ and the number $j \geq 0$ of the segment to which it belongs. It identifies its position in this segment from the round number in which it receives a message for the first time. Then a node outputs the line of length $k$ with its position in it.

The following lemma gives a lower bound on the time of topology recognition for lines, matching the performance of Algorithm `Line-Topology-Recognition`.

**Lemma 5.** *Let $D \geq 3$ be any integer, and let $c > 0$ be any real constant. For any line $T$, consider a labeling scheme LABEL($T$) of length at most $c$. Let TOPO be any algorithm that solves topology recognition for every line of diameter at most $D$ using the labeling scheme LABEL($T$). Then there exists a line of diameter at most $D$, for which TOPO must take time $\Omega(\log D)$.*

In view of the performance of Algorithm `Line-Topology-Recognition` and of Lemma 5, we have the following result.

**Theorem 6.** *The optimal time for topology recognition in the class of trees of maximum degree $\Delta = 2$ (i.e., lines) of diameter at most $D$, using a labeling scheme of length $O(1)$, is $\Theta(\log D)$.*

## 6 Conclusion

We established a tight bound $\Theta(\log \log \Delta)$ on the minimum length of labeling schemes permitting topology recognition in trees of maximum degree $\Delta$, and we proved upper and lower bounds on topology recognition time, using such short schemes. These bounds on time are almost tight: they leave a multiplicative gap smaller than any polynomial in $\Delta$. Closing this small gap is a natural open problem. Another interesting research topic is to extend our results to the class of arbitrary graphs. We conjecture that such results, both concerning the minimum length of labeling schemes permitting topology recognition, and concerning the time necessary for this task, may be quite different from those that hold for trees.

## References

1. Abiteboul, S., Kaplan, H., Milo, T.: Compact labeling schemes for ancestor queries. In: Proceedings of 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), pp. 547–556 (2001)
2. Chrobak, M., Gasieniec, L., Rytter, W.: Fast broadcasting and gossiping in radio networks. J. Algorithms **43**, 177–189 (2002)

3. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. ACM Trans. Algorithms **4**, 42 (2008)
4. Dereniowski, D., Pelc, A.: Drawing maps with advice. J. Parallel Distrib. Comput. **72**, 132–143 (2012)
5. Emek, Y., Fraigniaud, P., Korman, A., Rosen, A.: Online computation with advice. Theoret. Comput. Sci. **412**, 2642–2656 (2011)
6. Fraigniaud, P., Gavoille, C., Ilcinkas, D., Pelc, A.: Distributed computing with advice: information sensitivity of graph coloring. Distrib. Comput. **21**, 395–403 (2009)
7. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Communication algorithms with advice. J. Comput. Syst. Sci. **76**, 222–232 (2010)
8. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Tree exploration with advice. Inf. Comput. **206**, 1276–1287 (2008)
9. Fraigniaud, P., Korman, A., Lebhar, E.: Local MST computation with short advice. Theory Comput. Syst. **47**, 920–933 (2010)
10. Fusco, E., Pelc, A.: Trade-offs between the size of advice and broadcasting time in trees. Algorithmica **60**, 719–734 (2011)
11. Fusco, E., Pelc, A., Petreschi, R.: Topology recognition with advice. Inf. Comput. **247**, 254–265 (2016)
12. Gasieniec, L., Pagourtzis, A., Potapov, I., Radzik, T.: Deterministic communication in radio networks with large labels. Algorithmica **47**, 97–117 (2007)
13. Gasieniec, L., Peleg, D., Xin, Q.: Faster communication in known topology radio networks. Distrib. Comput. **19**, 289–300 (2007)
14. Gavoille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. J. Algorithms **53**, 85–112 (2004)
15. Glacet, C., Miller, A., Pelc, A.: Time vs. information tradeoffs for leader election in anonymous trees. In: Proceedings of 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016), pp. 600–609 (2016)
16. Ilcinkas, D., Kowalski, D., Pelc, A.: Fast radio broadcasting with advice. Theoret. Comput. Sci. **411**, 1544–1557 (2012)
17. Katz, M., Katz, N., Korman, A., Peleg, D.: Labeling schemes for flow and connectivity. SIAM J. Comput. **34**, 23–40 (2004)
18. Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. Distrib. Comput. **22**, 215–233 (2010)
19. Kowalski, D., Pelc, A.: Leader election in ad hoc radio networks: a keen ear helps. J. Comput. Syst. Sci. **79**, 1164–1180 (2013)
20. Nisse, N., Soguet, D.: Graph searching with advice. Theoret. Comput. Sci. **410**, 1307–1318 (2009)
21. Peleg, D.: Distributed Computing, a Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia (2000)

# Space-Time Tradeoffs for Distributed Verification

Rafail Ostrovsky[1], Mor Perry[2], and Will Rosenbaum[2(✉)]

[1] Department of Computer Science and Department of Mathematics, University of California, Los Angeles, Los Angeles, CA, USA
[2] School of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel
will.roenbaum@gmail.com

**Abstract.** Verifying that a network configuration satisfies a given boolean predicate is a fundamental problem in distributed computing. Many variations of this problem have been studied, for example, in the context of proof labeling schemes (PLS), locally checkable proofs (LCP), and non-deterministic local decision (NLD). In all of these contexts, verification time is assumed to be constant. Korman et al. [16] presented a proof-labeling scheme for MST, with poly-logarithmic verification time, and logarithmic memory at each vertex.

In this paper we introduce the notion of a $t$-PLS, which allows the verification procedure to run for super-constant time. Our work analyzes the tradeoffs of $t$-PLS between time, label size, message length, and computation space. We construct a universal $t$-PLS and prove that it uses the same amount of total communication as a known one-round universal PLS, and $t$ factor smaller labels. In addition, we provide a general technique to prove lower bounds for space-time tradeoffs of $t$-PLS. We use this technique to show an optimal tradeoff for testing that a network is acyclic (cycle free). Our optimal $t$-PLS for acyclicity uses label size and computation space $O((\log n)/t)$. We further describe a recursive $O(\log^* n)$ space verifier for acyclicity which does not assume previous knowledge of the run-time $t$.

## 1 Introduction

A fundamental problem in distributed computing is to determine if a network configuration satisfies some predicate. In the distributed setting, a network configuration is represented by an underlying graph, where each vertex represents a processor, edges represent communication links between processors, and each

R. Ostrovsky—Research supported in part by NSF grant 1619348, DARPA, US-Israel BSF grant 2012366, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The views expressed are those of the authors and do not reflect position of the Department of Defense or the U.S. Government.

M. Perry—Partially supported by Apple Graduate Fellowship.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 53–70, 2017.
https://doi.org/10.1007/978-3-319-72050-0_4

vertex has a state. For example, the state of every vertex can be a color, and the predicate signifies that the coloring is proper, i.e., that every edge has its endpoints colored differently. Processors learn about the network by exchanging messages along the edges. Some properties are local by nature and easy to verify, yet many natural problems—for example, testing if the network contains cycles—cannot be tested in less than diameter time, even if message size and local computational power are unbounded.

In order to cope with strong time lower bounds, Korman et al. introduced in [17] a computational model, called *proof-labeling schemes* (PLS), where vertices are given auxiliary global information in the form of *labels*. This auxiliary information may allow vertices to verify that a property is satisfied more efficiently than could be achieved without the aid of labels. Specifically, a PLS consists of two components: a *prover* and a *verifier*. The prover is an oracle that assigns labels to vertices. The verifier is a distributed algorithm that runs on the labeled configuration and outputs TRUE or FALSE at each vertex as a function of its state, its label, and the labels it receives. A PLS is *complete* if for every legal configuration (satisfying the predicate), the prover can assign labels such that all vertices output TRUE. The PLS is *sound* if for every illegal configuration (which does not satisfy the predicate) for every labeling, some vertex outputs FALSE.

Schemes for verifying a predicate are useful in many applications. One such application is checking the output of a distributed algorithm [3,12]. For example, if a procedure is meant to output a spanning-tree of the network, it may be useful to periodically verify that the output does indeed not contain cycles. If the original procedure which finds the spanning-tree can additionally produce labels, verification may be achieved substantially faster than diameter time required without the aid of labels. A simple procedure for checking the legality of the current state is very useful in the construction of self stabilizing algorithms [1,2,7,16]. Other applications include estimating the complexity of logics required for distributed run-time verification [12], establishing a general distributed complexity theory [11], and proving lower bounds on the time required for distributed approximation [8]. Local verification was recently applied in the design and analysis of software defined networks (SDN) in [18].

Distributed verification has been formalized in various models to suit its myriad applications. These models include proof-labeling schemes (PLS) [17], locally checkable proofs (LCP) [13], and non-deterministic local decision (NLD) [11]. We refer the reader to [9] for a detailed comparison of these models. All three of these models are local in the sense that verification requires a constant number of rounds, independent of the size of the graph. PLS differs from LCP and NLD in that verification in (traditional) PLS occurs in a single communication round, while the LCP and NLD models allow verification in a fixed constant number of rounds. While a fast procedure is certainly a desirable feature in verification algorithms, it may be the case that other computational resources—space or communication—must also be considered. For example, in the case of PLS, deterministically verifying a sub-graph is acyclic requires labels of size $\Omega(\log n)$

per vertex [17]. However, specifying a sub-graph only requires $O(\Delta)$ space (the maximum degree of a vertex) per vertex. Thus, if we restrict attention to local verification algorithms, the space requirement to store labels may be unboundedly larger than the space required to specify the instance.

Korman et al. [16] presented a PLS for minimum spanning-tree with poly-logarithmic verification time and logarithmic memory at each vertex. In the present work we also consider super-constant time verification and address tradeoffs between computational resources in distributed verification algorithms: label size, communication, computation space, and time. Specifically, we address the following questions: If verification algorithms are allowed to run in super-constant time, can labels be significantly shorter? What are the tradeoffs between label size and verification time? Can verification be achieved using (per processor) space which is linear in the label size? We focus on the acyclicity problem and prove that labels can indeed be shortened by a factor of $t$—the run-time of the algorithm—compared to constant-round verification. Moreover, computation space for each vertex can be made linear in the label size. Note that in this model it does not trivially hold that each message contains exactly one label, since in each round every vertex receives a (potentially different) label from each neighbor, and the scheme should specify the message to be sent in the following round. We show that in our schemes messages are small enough so that the total communication is the same as in one-round verification.

### 1.1 Our Contributions

In this paper we consider proof-labeling schemes with super-constant verification time, and analyze tradeoffs between time, label size, message size, and computation space. Many of the results presented here were announced without proof in [5]. In Subsect. 3.1, we describe a universal scheme which can verify any property $\mathcal{P}$. Suppose $G_s$, with $n$ vertices, $m$ edges, and each state can be represented using $s$ bits. Then for every $t \in O(\mathrm{diam}(G_s))$, our scheme verifies $\mathcal{P}$ in $t$ rounds using labels and messages of size $O((ns + \min\{n^2, m \log n\})/t)$. For $t = 1$ this is the known universal scheme [4,13,17]. When $t \in \Omega(n)$, we obtain labels and messages of size $O(s + \min\{n, (m/n) \log n\})$. Overall, labels are significantly smaller, and total communication is the same. Subsect. 3.2 proves a general lower bound technique for label size of $t$-round schemes.

In Sect. 4 we consider the problem determining if a graph is acyclic. Using the lower bound technique of Subsect. 3.2, we prove in Subsect. 4.1 that labels of size $\Omega((\log n)/t)$ are required for the ACYCLIC problem. Subsect. 4.2 shows that this lower bound is tight. Our scheme for ACYCLIC additionally uses optimal space and messages of size $O((\log n)/t)$. In particular, by taking $t$ to be a sufficiently large constant, our upper bound (along with the $\Omega(\log n)$ lower bound for ACYCLIC in [17]) implies separation between the PLS and LCP models for acyclicity (see [9]). The verifier for ACYCLIC assumes that vertices are given some truthful information about the round number, for example, by being told when (a multiple of) $t$ rounds have elapsed. We show that such information is necessary for *any* super-constant and sub-linear time distributed algorithm.

In Subsect. 4.3, we describe a recursive scheme for ACYCLIC which uses space $O(\log^* n)$ and constant communication per vertex per round. The recursive verifier runs in time $O(n)$ in the worst case, but there are always correct labels which will be accepted in time $O(\log \mathrm{diam}(G))$. We note that in order to break the logarithmic space barrier, our schemes in Subsects. 4.2 and 4.3 crucially do not rely upon unique identifiers for the vertices. Conversely, the lower bounds of Subsects. 3.2 and 4.1 hold for a stronger model where vertices have unique identifiers, and labels may depend on the unique identifiers.

### 1.2 Related Work

Distributed verification has been studied extensively. It was studied and used in the design of self stabilizing algorithms, first in [1], where the notion of local detection was introduced, and recently in [16], where a super-constant time verification scheme was presented. Both papers use verification in the design of a self stabilizing algorithm for constructing a minimum spanning-tree. Verification has also received attention of its own. For example, [15] presented tight bounds for minimum spanning-tree verification. In [17], Korman et al. formalized the concept of local verification and introduced the notion of proof-labeling schemes. In their paper, verification is defined to use one communication round, and among other results they show a $\Theta(\log n)$ bound on the complexity (label size and communication) for ACYCLIC. Recently, [4] suggested using randomization in order to break the lower bounds of deterministic schemes, and among other results they show a $\Theta(\log \log n)$ bound on the communication complexity of acyclicity. In this paper, we show that if we use super-constant verification time, we can break the lower bound of space consumption (label size and computation space), while the total amount of communication is the same as in one deterministic verification round. Proof-labeling schemes with constant, greater than one, verification time was studied in [13], and with super-constant verification time was presented in [16]. In [10], the authors consider verification of acyclicity and related problems in various models for directed graphs.

The question of what properties can be verified using a constant verification time was studied in [11], and several complexity classes were presented, including LD—local decision—which includes all properties that can be decided using constant number of rounds and no additional information, and NLD—nondeterministic local decision—which includes all properties that can be decided in a constant number of rounds with additional information in the form of a certificate given to each vertex. While NLD and PLS are closely related, they differ in that NLD certificates are independent of vertex identifiers. Since PLS labels may depend on vertex identifiers, there is a PLS for every sequentially decidable property on ID based networks, while not all sequentially decidable properties are in NLD. Our lower bounds in Subsects. 3.2 and 4.1 allow labels to depend on unique vertex identifiers, so our arguments give identical lower bounds for certificate sizes in the weaker NLD model. Nonetheless, the schemes for ACYCLIC in Subsects. 4.2 and 4.3 do not require unique identifiers.

Awerbuch and Ostrovsky describe a $\log^* n$-space distributed acyclicity verifier in [2]. Our scheme described in Sect. 4.3 achieves the same space usage per node, but improves on the algorithm of [2] in several ways. The worst-case runtime of our acyclicity verifier is $O(n)$, whereas that in [2] requires time $O(n \log^2 n)$. Further, in our scheme there are always correct labels which are accepted in time $O(\log n)$. This runtime nearly matches the $\Omega((\log n)/\log^* n)$ time lower bound implied by Theorem 7. We leave it as an open question if it is possible to verify ACYCLIC using constant space and worst case runtime $O(\log n)$.

## 2 Model and Definitions

### 2.1 Computational Framework

A ***graph configuration*** $G_s$ consists of an underlying graph $G = (V, E)$, and a state assignment function $\varphi : V \to S$, where $S$ is a state space. The state of a vertex includes all of its local information. It may include the vertex's identity (in an ID based configuration), the weight of its adjacent edges (in a weighted configuration), or the result of an algorithm executed on the graph, for example, its color according to a coloring algorithm.

In a proof-labeling scheme, an oracle assigns labels $\ell : V \to L$. Verification is performed by a distributed algorithm on the labeled configuration in synchronous rounds. In each round every vertex receives messages from all of its neighbors, performs local computation, and sends a message to all of its neighbors. At the beginning of each round, a vertex scans its messages in a streaming fashion, and the ***computational space*** is the maximum space required by a vertex in its local computation. Each vertex may send different messages to different neighbors in a round. When a vertex halts, it outputs TRUE or FALSE. If the vertex labels contain unique identifiers, then we require that an algorithm has the same output for all legal assignments of unique IDs.

### 2.2 Proof-Labeling Schemes and $t$-PLS

We start with a short description of proof-labeling schemes (PLS) as introduced in [17]. Given a family $\mathcal{F}$ of configurations, and a boolean predicate $\mathcal{P}$ over $\mathcal{F}$, a PLS for $(\mathcal{F}, \mathcal{P})$ is a mechanism for deciding $\mathcal{P}(G_s)$ for every $G_s \in \mathcal{F}$. A PLS consists of two components: a ***prover*** $\mathbf{p}$, and a ***verifier*** $\mathbf{v}$. The prover is an oracle which, given any configuration $G_s \in \mathcal{F}$, assigns a bit string $\ell(v)$ to every vertex $v$, called the ***label*** of $v$. The verifier is a distributed algorithm running concurrently at every vertex. The verifier $\mathbf{v}$ at each vertex outputs a boolean. If the outputs are TRUE at all vertices, $\mathbf{v}$ is said to ***accept*** the configuration, and otherwise (i.e., $\mathbf{v}$ outputs FALSE in at least one vertex) $\mathbf{v}$ is said to ***reject*** the configuration. For correctness, a proof-labeling scheme $(\mathbf{p}, \mathbf{v})$ for $(\mathcal{F}, \mathcal{P})$ must be (1) ***complete*** and (2) ***sound***. Formally, for every $G_s \in \mathcal{F}$, we say $(\mathbf{p}, \mathbf{v})$ is

1. ***complete*** if whenever $\mathcal{P}(G_s) =$ TRUE then, using the labels assigned by $\mathbf{p}$, the verifier $\mathbf{v}$ accepts $G_s$, and

2. ***sound*** if whenever $\mathcal{P}(G_s) = \text{FALSE}$ then, for every label assignment, the verifier $\mathbf{v}$ rejects $G_s$.

The ***verification complexity*** of a proof-labeling scheme $(\mathbf{p}, \mathbf{v})$, according to [17], is the maximal label size—the maximal length of a label assigned by the prover $\mathbf{p}$ on a legal configuration (satisfying $\mathcal{P}$). A PLS is defined to use one verification round, in which neighbors exchange labels. In this case, label size and message size are the same.

In this paper we consider proof-labeling schemes with more than one verification round, in particular it can use super-constant time, and hence we define the ***message size*** of the scheme $(\mathbf{p}, \mathbf{v})$ to be the largest message a vertex sends during the execution of $\mathbf{v}$ on a legal configuration with the labels assigned by $\mathbf{p}$. We denote a proof-labeling scheme with $t$-round verification by $t$-PLS.

## 3 General Space-Time Tradeoff Results

In this section we give general results for label size reduction and message size in a $t$-PLS. The idea is to take a 1-PLS, and break it into smaller ***shares*** where vertices are assigned only a single share of the original label. We refer to this technique as ***label sharing***. In particular, we present a universal scheme and provide a tool for obtaining lower bounds. We first observe that if there exists a PLS for $(\mathcal{F}, \mathcal{P})$ with label size $\kappa$ (and hence, message size $\kappa$), then there exists a $t$-PLS for $(\mathcal{F}, \mathcal{P})$ with label size $\kappa$ and message size $\kappa/t$. Indeed, vertices can communicate their $\kappa$-bit label in $t$ different shares of size $\kappa/t$, where the original label is simply the concatenation of the shares. In the universal scheme described below, the oracle assigns each vertex only a single share. Each vertex then reconstructs the original (1-PLS) labeling from the shares received from neighbors in $t$ communication rounds.

### 3.1 Universal $t$-PLS

A ***universal scheme*** is a scheme that verifies every sequentially decidable property. In this subsection we assume that every vertex has an identifier, and identifiers in the same configuration are pairwise distinct. We give an upper bound on the label and message size of a universal scheme that uses $t$ communication rounds.

**Theorem 1.** *Let $\mathcal{F}$ be a family of configurations with states set $S$ and diameter at least $D$, let $\mathcal{P}$ be a boolean predicate over $\mathcal{F}$ and suppose that every state in $S$ can be represented using $s$ bits. For every $t \in \Omega(D)$ there exists a $t$-PLS for $(\mathcal{F}, \mathcal{P})$ with label and message size $O((ns + \min\{n^2, m \log n\})/t)$ where $n$ is the number of vertices, and $m$ is the number of edges in the graph.*

In the proof of this theorem we use a known universal PLS [4,13,17]. Labels consist of the entire representation of the graph configuration. Nodes

then verify that they have the same representation, and that it is consistent with its local view. Finally, they verify individually that the label represents a legal configuration. Since every configuration can be represented using $O(ns + \min\{n^2, m \log n\})$ bits—by listing the state of each vertex and an adjacency matrix or an edge list—this is the label (and message) size of this scheme.

The idea of the universal $t$-PLS is to disperse the configuration representation into shares such that each vertex can collect the purported graph configuration from its $t$-neighborhood. The details and the formal proof appear in the full version of this paper [6].

### 3.2 Lower Bound Tool

We start with some definitions. Although we consider only networks represented by undirected graphs, we will define an orientation on an edge to indicate a specific ordering of its endpoints. We denote by $H(e)$ the head of a directed edge $e$, and by $T(e)$ the tail of $e$.

**Definition 2 (Edge Crossing).** *Let $G = (V, E)$ be a graph, and $e_1, e_2 \in E$ be two directed edges. The* edge crossing *of $e_1$ and $e_2$ in $G$, denoted by $C(e_1, e_2, G)$, is the graph obtained from $G$ by replacing $e_1$ and $e_2$, by the edges $(T(e_1), H(e_2))$ and $(T(e_2), H(e_1))$.*

Edge crossings were used many times before, and were formalized as a tool for proving lower bounds of verification complexity in [4]. We now show how to use edge crossing in order to prove lower bounds for label size of $t$-PLS.

**Definition 3 (Edge $k$ -neighborhood).** *Let $G = (V, E)$ be a graph, and $e = (u, v) \in E$. The $k$-**neighborhood** of $e$ in $G$, denoted by $N_k(e, G)$, is the subgraph $(V', E')$ of $G$ satisfying*

*1. $w \in V'$ if and only if $w \in V$ and $\min(\text{dist}(w, u), \text{dist}(w, v)) \leq k$, and*
*2. $e' \in E'$ if and only if $e' \in E \cap (V' \times V')$.*

**Proposition 4.** *Let $(\mathbf{p}, \mathbf{v})$ be a deterministic $t$-PLS for $(\mathcal{F}, \mathcal{P})$ with label size $|\ell|$. Suppose that there is a configuration $G_s \in \mathcal{F}$ which satisfies $\mathcal{P}$ and contains $r$ directed edges $e_1, \ldots, e_r$, whose $t$-neighborhoods $N_t(e_1, G_s), \ldots, N_t(e_r, G_s)$ are pairwise disjoint, contain $q$ vertices each, and there exist $r$ state preserving isomorphisms $\sigma_i : V(N_t(e_1, G_s)) \to V(N_t(e_i, G_s))$ for $i = 1, \ldots, r$ such that $\sigma_i(H(e_1)) = H(e_i)$ and $\sigma_i(T(e_1)) = T(e_i)$. If $|\ell| < (\log r)/q$, then there exist $i, j$ with $1 \leq i < j \leq r$ such that every connected component of $C(e_i, e_j, G_s)$ is accepted by $(\mathbf{p}, \mathbf{v})$.*

*Proof.* Let $(\mathbf{p}, \mathbf{v})$ and $G_s$ be as described above, and assume that $|\ell| < (\log r)/q$. Consider a collection $\{\sigma_i : V(N_t(e_1, G_s)) \to V(N_t(e_i, G_s)), i = 1, \ldots, r\}$ of $r$ state preserving isomorphisms, such that $\sigma_i(H(e_1)) = H(e_i)$ and $\sigma_i(T(e_1)) = T(e_i)$. Order the vertices of $N_t(e_1, G_s)$ arbitrarily. For every $i$, consider the concatenation of labels given by $\mathbf{p}$ to the vertices of $N_t(e_i, G_s)$, in the order induced

by the ordering of $N_t(e_1, G_s)$ and $\sigma_i$. Denote this concatenated string $L_i$. By label size assumption, it holds that $|L_i| < \log r$ for every $i$, and thus there are less than $r$ different options for $L_i$. Therefore, by the pigeonhole principle, there are $i \neq j$ such that $L_i = L_j$. Denote $C(e_i, e_j, G_s)$ by $G'_s$, and consider the labels provided by $\mathbf{p}$ to $G_s$. For every vertex $v \notin N_t(e_i, G_s) \cup N_t(e_j, G_s)$, its $t$-neighborhood is the same in $G_s$ and in $G'_s$. $N_t(e_i, G_s)$ and $N_t(e_j, G_s)$ are disjoint, isomorphic, and have the same states and labels according to some isomorphism which maps $H(e_i)$ to $H(e_j)$ and $T(e_i)$ to $T(e_j)$. Thus, for every vertex $v \in N_t(e_i, G_s) \cup N_t(e_j, G_s)$, its $t$-neighborhood in $G_s$ is the same as in $G'_s$. Since the output of the verifier $\mathbf{v}$ at each vertex in $G_s$ is only a function of the states and labels at its $t$-neighborhood, if the output of $\mathbf{v}$ in $G_s$ is TRUE at all vertices, then the output of $\mathbf{v}$ in every connected component of $G'_s$ must be TRUE, and the proposition follows.

The following theorem, which is a consequence of Proposition 4, is the tool we use to prove lower bounds of label size in a $t$-PLS.

**Theorem 5.** *Let $\mathcal{F}$ be a family of configurations, and let $\mathcal{P}$ be a boolean predicate over $\mathcal{F}$. Suppose that there is a configuration $G_s \in \mathcal{F}$ which satisfies*

1. *$\mathcal{P}(G_s) =$ TRUE,*
2. *$G_s$ contains $r$ directed edges $e_1, \ldots, e_r$, whose $t$-neighborhoods $N_t(e_1, G_s), \ldots, N_t(e_r, G_s)$ are pairwise disjoint, contain $q$ vertices each, and there exist $r$ state preserving isomorphisms $\{\sigma_i : V(N_t(e_1, G_s)) \to V(N_t(e_i, G_s)), i = 1, \ldots, r\}$ such that $\sigma_i(H(e_1)) = H(e_i)$ and $\sigma_i(T(e_1)) = T(e_i)$, and*
3. *for every $i \neq j$, there exists a connected component $H_s$ of $C(e_i, e_j, G_s)$ such that $\mathcal{P}(H_s) =$ FALSE.*

*Then the label size of any $t$-PLS for $(\mathcal{F}, \mathcal{P})$ is $\Omega((\log r)/q)$.*

## 4 Acyclicity

In this section we focus on the acyclicity property, and give tight $t$-PLS lower and upper bounds. The lower bounds of Subsect. 4.1 hold in the computational model where vertices have unique identifiers, and the labels are allowed to depend on the ID of a vertex. The upper bounds presented in Subsects. 4.2 and 4.3 still apply in a weaker computational model where vertices do not have unique IDs.

**Definition 6 (Acyclicity).** *Let $\mathcal{F}$ be the family of all connected graphs. Given a graph configuration $G_s \in \mathcal{F}$,* ACYCLIC$(G_s) =$ TRUE *if and only if the underlying graph $G$ is cycle free.*

### 4.1 Lower Bound for ACYCLIC

**Theorem 7.** *Every scheme which verifies* ACYCLIC *in $t$ communication rounds requires labels of size $\Omega((\log n)/t)$.*

*Proof.* We will show a configuration as described in Theorem 5, with $r = \Omega(n/t)$ and $q = O(t)$, to derive the stated lower bound on label size of any scheme that verifies ACYCLIC. Let $G_s$ be the $n$-vertex path $v_0 - v_1 - \cdots - v_{n-1}$ where all states are the empty string. Obviously ACYCLIC$(G_s) =$ TRUE. Let $r = \lfloor n/(2t+2) \rfloor - 1$, and consider the set $\{e_i = (v_{(2t+2)i}, v_{(2t+2)i+1}) \mid 1 \leq i \leq r\}$ of $r$ directed edges. Each $N_t(e_i, G_s)$ contains exactly $2t+2$ vertices, and thus $q = 2t+2$. Every pair of $t$-neighborhoods $N_t(e_i, G_s)$ and $N_t(e_j, G_s)$, for $i \neq j$, is disjoint since the distance between $e_i$ and $e_j$ is at least $2t+1$. For every $i < j$, $C(e_i, e_j, G_s)$ contains exactly two connected components. One of them is the cycle $H_s = v_{qi+1} - v_{qi+2} - \cdots - v_{qj} - v_{qi+1}$ where all its edges are marked. By definition, $\mathcal{P}(H_s) =$ FALSE. Hence, the conditions of Theorem 5 are satisfied, and the lower bound follows.

### 4.2 Upper Bound for ACYLCLIC

In this section, we describe a $t$-PLS for ACYCLIC which matches the lower bound presented in Theorem 7.

**Theorem 8.** *Suppose $G = (V, E)$ is a graph with diameter $D$. For every $t \leq \min\{\log n, D\}$, there exists an $O(t)$-PLS for* ACYCLIC *with label and messages of size $O((\log n)/t)$. Further, the verifier* **v** *uses space of size $O((\log n)/t)$.*

**Remark 9.** *In this subsection, we assume that each vertex has access to some means of deciding (correctly) when $t$ communication rounds have elapsed. This can be achieved either by allowing each vertex a $\log t$ bit counter, or by giving each vertex access to an oracle which alarms when (an integer multiple of) $t$ rounds have elapsed. We discuss the necessity of this assumption in Subsect. 4.3.*

The following scheme can be used to verify that the graph contains no cycles using labels of size $O(\log n)$ in a single round. The label of a vertex $v$ consists of an integer $d(v)$ which encodes the distance from $v$ to a root vertex (which has $d(v) = 0$). Vertices verify the correctness of the labels in a single communication round. If $v$ satisfies $d(v) = 0$ (i.e., $v$ is a root), then it accepts the label if all of its neighbors $w$ satisfy $d(w) = 1$. If $v$ satisfies $d(v) \neq 0$ then $v$ verifies that $v$ has exactly one neighbor $u$ with $d(u) = d(v) - 1$ while all other neighbors $w$ satisfy $d(w) = b(v) + 1$. This scheme is used, for example, in [2,3,14]. The correctness of the scheme is a consequence of the following definition and lemma.

**Definition 10.** *Suppose $G = (V, E)$ is a graph and $L = \{0, 1, \ldots, s-1\}$ with $s \geq 3$. We call function $\ell : V \to L$ an **$s$-cyclic labeling** of $G$ if for every $v \in V$, $v$ has at most one neighbor $P(v)$—the **parent** of $v$—such that $\ell(P(v)) \equiv \ell(v) - 1 \mod s$, while the $v$'s other neighbors $w$ satisfy $\ell(w) \equiv \ell(v) + 1 \mod s$.*

**Remark 11.** *An $s$-cyclic labeling induces an orientation on $G$ where an edge $(u, v)$ is oriented such that $u = P(v)$. That is, each edge is oriented away from the parent.*

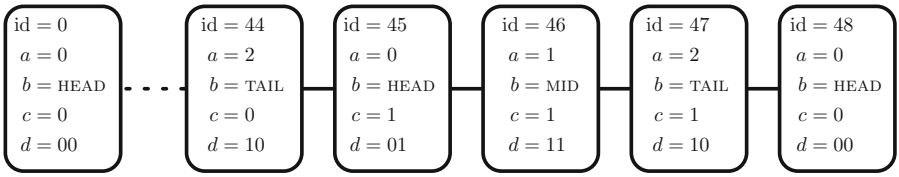



**Fig. 1.** Acyclicity labels for a graph consisting of a path rooted at its left endpoint. We have given the nodes identifiers $0, 1, \ldots$ from left to right, although the labeling need not include the id of the vertices. For this configuration, the orientation labels $a(v)$ simply count the distance from $v$ to the root (with id 0) modulo 3. The nodes with ids 45 6, and 47 form a single block, whose head (45) and tail (47) are indicated by the corresponding block labels. The color of this block is 1 because it is the 15th block from the root ($45/3 = 15$), and $15 \equiv 1 \mod 2$. Finally, the concatenation of the distance labels in this block is $d(47)d(46)d(45) = 101101$, which encodes the distance of the block's head to the root (45) in binary.

**Lemma 12.** *Suppose $G = (V, E)$ is a connected graph and $\ell$ an $s$-cyclic labeling. Then either $G$ is acyclic or $G$ contains a unique cycle of length $k$, where $s$ divides $k$. Further, if $G$ contains a cycle, $C$, then $C$ is an oriented cycle in the orientation induced by $\ell$, and all oriented paths in $G$ are oriented away from vertices in $C$.*

*Proof.* Suppose $C = (v_0, v_1, \ldots, v_{k-1})$ is a cycle in $G$. In the orientation described in Remark 11, every vertex has in-degree at most 1. Let $\deg_{in}(v_i)$ denote the in-degree of $v_i$ in $C$ and similarly $\deg_{out}(v_i)$ is $v_i$'s out-degree in $C$. Then $\deg_{in}(v_i) - \deg_{out}(v_i) \leq 0$ for all $v_i$. However, we must have $\sum_i \deg_{in}(v_i) - \deg_{out}(v_i) = 0$, implying that in fact $\deg_{in} v_i = \deg_{out}(v_i) = 1$ for all $i$. Thus, $C$ is an oriented cycle. As a consequence, for all $i$, either $\ell(v_i) \equiv \ell(v_{i+1}) + 1 \mod s$ or $\ell(v_i) \equiv \ell(v_{i+1}) - 1 \mod s$. In the former case, we have $\ell(v_{k-1}) - \ell(v_0) \equiv k \equiv 0 \mod s$, implying that $s$ divides $k$. In the latter case, $\ell(v_{k-1}) - \ell(v_0) \equiv -k \equiv 0 \mod s$, and the desired result holds.

Since every vertex $v_i \in C$ has in-degree 1 in $C$, all edges that leave $C$ must be oriented away from vertices in $C$. Similarly, any path $w_0, w_1, \ldots, w_j$ with $w_0 \in C$ and $w_i \notin C$ for $i \geq 1$ must be oriented away from $C$. Thus no such path may lead to another cycle $C'$, nor could another cycle $C'$ share a path with $C$. Thus since $G$ is connected $C$ must the unique cycle.

To achieve labels of length $O((\log n)/t)$ for ACYCLIC, we simulate the "distance-to-root" scheme described above. The idea is to break the $O(\log n)$-bit labels indicating the distance to the root into shares of size $O((\log n)/t)$. Unlike the universal scheme described in Subsect. 3.1, vertices do not reconstruct the $(\log n)$-bit distance-to-root labels directly, but check the labeling is correct distributively. Thus the verifier $\mathbf{v}$ only uses space linear in the label size.

Formally, for a vertex $v$, an ***acyclicity label*** consists of:

- an ***orientation label*** $a(v) \in \{0, 1, 2\}$ which defines an orientation on edges away from the root of the tree,

- a ***block label*** $b(v) \in \{\text{HEAD}, \text{MID}, \text{TAIL}\}$ which indicates $v$'s position within a block,
- a ***block color*** $c(v) \in \{0, 1\}$, and
- a ***distance label*** $d(v) \in \{0,1\}^{(\log n)/t}$ which encodes a share of a distance to the root.

See Fig. 1 for an example of correctly formed labels. It is clear that an acyclicity label can be recorded in $O((\log n)/t)$ bits. The semantics of acyclicity labels are described below.

**Correct orientation labels.** The orientation labels $a(v)$ are correct if every $v \in V$ has at most one neighbor $P(v)$—the ***parent*** of $v$—such that $a(P(v)) \equiv a(v) - 1 \mod 3$. The remaining neighbors $w$ of $v$—$v$'s ***children***—satisfy $a(w) \equiv a(v) + 1 \mod 3$. If $P(v) = \varnothing$, we call $v$ a ***root***. Correct orientation labels induce an orientation on $G$ where the oriented edges $(v, w)$ satisfy $a(w) \equiv a(v) + 1 \mod 3$. Thus, edges are oriented away from roots (if any).

**Correct block labels.** Block labels must be assigned in the following manner

1. $b(v) = \text{HEAD}$ if and only if either $P(v) = \varnothing$ or $b(P(v)) = \text{TAIL}$
2. $b(v) = \text{TAIL}$ if and only if there exists an oriented path of length $t$, $v_0, v_1, \ldots, v_{t-1} = v$ such that $b(v_0) = \text{HEAD}$. We refer to such a path as a ***block***.
3. In all other cases, $b(v) = \text{MID}$.
4. For every $v$, there exists an oriented path $w_0, w_1, \ldots, w_{k-1} = v$ of length $k < t$ such that $b(w_0) = \text{HEAD}$.

**Definition 13.** *Let $B = (v_0, v_1, \ldots, v_{t-1})$ be a block. We define the **value** of $B$, denoted $D(B)$, to be the integer whose binary expansion is the concatenation $d(v_{t-1})d(v_{t-2}) \cdots d(v_0)$. That is, $v_0$ holds the least significant bits of $D(B)$, while $v_{t-1}$ holds the most significant bits. If $B' = (w_0, w_1, \ldots, w_{t-1})$ is another block, we say that $B$ is the **parent** of $B'$ and $B'$ is a **child** of $B$ if $P(w_0) = v_{t-1}$. If there exists $i$ such that $v_i = w_i$, we say that $B$ and $B'$ **overlap**.*

**Correct block coloring.** The block coloring $c$ is correct if

1. for every block $B$ and $v, w \in B$ we have $c(v) = c(w)$, and
2. for every blocks $B, B'$ such that $B$ is the parent of $B'$, and $v \in B$, $w \in B'$, we have $c(v) \neq c(w)$.

**Correct distance labels.** The distance labels $d$ are correct if

1. for every block, $B = (v_0, v_1, \ldots, v_{t-1})$, $D(B) = 0$ if and only if $v_0$ is a root, and
2. for every pair of blocks $B$ and $B'$ with $B$ the parent of $B'$, we have $D(B') = D(B) + t$.

**Definition 14 (Correct acyclicity labeling).** *Suppose $\ell$ is a family of acyclicity labels for a graph $G = (V, E)$. We say that the family $\ell$ is **correct** if $a$, $b$, $c$, and $d$ are correct orientation labels, correct block labels, correct block colorings, and correct distance labels as described above.*

**Remark 15.** *If blocks $B = (v_0, \ldots, v_{t-1})$ and $B' = (w_0, \ldots, w_{t-1})$ overlap, then we must have $w_0 = v_0$ and $D(B) = D(B')$. The first equality holds because each vertex $v_i$ has at most one parent, so if $w_i = v_i$ we must have $w_j = v_j$ for $0 \leq j \leq i$. The second equation holds because either $B$ and $B'$ contain a root, in which case $D(B) = D(B') = 0$, or there is a $B''$ which is the parent of both $B$ and $B'$. In the latter case, $D(B) = D(B'') + t = D(B')$.*

**Proposition 16.** *Let $G = (V, E)$ be a graph. Then $G$ is acyclic if and only if it admits a correct labeling $\ell$.*

*Proof.* If $G$ is acyclic, then we can form labels $\ell$ in the following way. Choose an arbitrary vertex $u$ to be the root. For all $v$ define $d'(v) = \text{dist}(v, u)$ (the length of the unique path from $v$ to $u$), and take $a(v) = d'(v) \mod 3$. Define $b(v)$ by $b(v) = \text{HEAD}$ if $d'(v) \equiv 0 \mod t$, $b(v) = \text{TAIL}$ if $d'(v) \equiv -1 \mod t$, and $d(v) = \text{MID}$ otherwise. Finally, assign distance labels $d(v)$ in such a way that in each block $B$ with first element $v_0$, $D(B) = d'(v_0)$. It is easy to verify that these labels $\ell$ constructed in this way will satisfy all the provisions of Definition 14.

Conversely, suppose $G$ admits a correct family of acyclicity labels. Suppose towards a contradiction that $C = (w_0, w_1, \ldots, w_{k-1})$ is a cycle. Since the orientation labels $a(v)$ are correct (hence form a 3-cyclic labeling), $C$ must be an oriented cycle (as in the proof of Lemma 12). The final provision in the correctness of $b$ and the fact that each vertex $w_i$ has a unique parent guarantee some $w_i$ must have $b(w_i) = \text{HEAD}$. Without loss of generality, assume that $b(w_0) = \text{HEAD}$, and let $B_0$ be the block containing $w_0$ and contained in $C$. Inductively define blocks $B_1, B_2, \ldots \subseteq C$ such that $B_{i+1}$ is a child of $B_i$. By the pigeonhole principle, we must have $B_i = B_j$ for some $i < j$. However, the correctness of the distance labels implies that $D(B_i) < D(B_{i+1}) < \cdots < D(B_j) = D(B_i)$, a contradiction.

In order to prove Theorem 8, by Proposition 16, it suffices to show there is a verifier $\mathbf{v}$ for acyclicity labels which runs in time $O(t)$ using messages and memory of size $O((\log n)/t)$. Verification of the correctness of the orientation labels $a$, block coloring $c$, and conditions 1 and 3 in the correctness of the block labels $b$ can be accomplished in a single communication round with constant communication. Thus, we must verify conditions 2 and 4 in the correctness of the block labels as well as the correctness of distance labels.

After the initial sharing of labels with neighbors in the first round, the verification algorithm $\text{VERIFY}(v, a, b, c, d)$ continues as follows (see Algorithm 1 for pseudo-code). For $t - 1$ steps, each vertex relays the message from its parent to all of its children. At the end of $t$ rounds, each vertex verifies that at some point, it received a message from a head vertex. If a vertex $v$ received a message from a root vertex, it verifies that $d(v) = 0$. Otherwise, let $b(w)$, $c(w)$, and $d(w)$ be labels received by $v$ in the $t$-th round. Then $v$ checks that $b(w) = b(v)$, $c(w) \neq c(v)$. The block heads increment the distance labels $d(w)$ $t$ times, sending carry bits (if any) to their children. When children receive carry bits, they increment their $d(w)$'s accordingly, sending further carry bits to their children. After this incrementation procedure, vertex $v$ verifies that the incremented $d(w)$'s satisfy $d(v) = d(w)$.

**Algorithm 1.** VERIFY$(v, a, b, c, d)$: Verifies correctness of acyclicity labels.

```
1: send a(v), b(v), and c(v) to all neighbors
2: verify correctness of a and c, and conditions 1 and 3 in correctness of b
3: HEAD_CHECK ← false
4: if b(v) = TAIL then
5:    IS_ZERO ← true
6: end if
7: for i = 1 to t-1 do
8:    M ← (b(w), c(w), d(w)) or ∅ received from P(v)
9:    if b(w) = HEAD then
10:      HEAD_CHECK ← true
11:   end if
12:   if b(v) = TAIL then
13:      if d(w) ≠ 0 then
14:         IS_ZERO ← FALSE
15:      end if
16:      if i = t − 1 then
17:         assert: b(w) = HEAD
18:      end if
19:   end if
20:   send M to all children {if v is a leaf, ignore}
21: end for
22: if M = ∅ then
23:   assert: d(v) = 0 {head of v's block is root}
24: else
25:   for i = 1 to t do
26:      INCREMENT(d(w), |d(w)|, 1)
27:   end for
28:   assert: b(w) = b(v)
29:   assert: c(w) ≠ c(v)
30:   assert: d(w) = d(v)
31:   if b(v) = TAIL then
32:      assert: IS_ZERO = false
33:   end if
34: end if
35: assert: HEAD_CHECK = true
```

**Lemma 17.** *Let $\ell$ be a family of acyclicity labels on a graph $G = (V, E)$. Then $\ell$ is correct if and only if every vertex $v$ accepts in Algorithm 1.*

*Proof.* By induction, each vertex receives the message from its (unique) $i$-th ancestor in the $i$-th communication round. Therefore, every tail accepts at lines 16–18 if and only if every tail is at (oriented) distance $t-1$ from a head. Similarly, every vertex $v$ is at (oriented) distance $i_v < t$ from a head if and only if it accepts at line 35 (see lines 9–11). Thus, the block labels are correct if and only if every vertex accepts at lines 2, 17, and 35.

Note that $b(w) = \varnothing$ if and only if the head of the block containing $v$ is a root. Thus, every vertex accepts at line 23 if and only if all blocks $B$ containing a root satisfy $D(B) = 0$. Conversely, if $B$ does not contain a root, then by the assertion at line 32 (and the check at lines 13–15), then $D(B) \neq 0$. Thus the checks at lines 23 and 32 are satisfied if and only if condition 1 in the correctness of distance labels is satisfied.

Suppose block $B = (w_0, \ldots, w_{t-1})$ is the parent of $B' = (v_0, \ldots, v_{t-1})$, then the distance label received by each $v_i$ is $d(w_i)$. Thus, after incrementing the labels $d(w_0)d(w_1)\cdots d(w_{t-1})$ $t$ times, the incremented labels will have value $D(B) + t$. Therefore, all vertices in $B'$ accept at line 31 if and only if $D(B') = D(B) + t$, if and only if condition 2 of correct distance labels is satisfied.

*Proof (of Theorem 8).* Lemma 17 implies that the VERIFY routine (Algorithm 1) is a correct verifier for acyclicity labels. Thus we must only argue that VERIFY

achieves the claimed time, space, and communication bounds. In each communication round, each vertex broadcasts a single label (in line 20) or a single bit (in INCREMENT) to its neighbors. Thus, the communication in each round is $O((\log D)/t)$ per edge. In each iteration of the algorithm, each vertex stores at most a constant number of labels, hence the memory usage is $O((\log D)/t)$ as well. Finally, the overall run-time is $3t$. The label sending procedure in lines 7–21 is accomplished in $t$ rounds, while the incrementation procedure in lines 25–7 requires at most $2t$ rounds: $t$ rounds where the head vertices increment, and another $t$ to propagate carries. In particular, the run-time is $O(t)$.

### 4.3 Recursive Acyclicity Checking

The scheme described in Subsect. 4.2 gives asymptotically optimal label size for $t \leq \log n$. Further, the communication per round and local memory usage is linear in the label size. However, the scheme above crucially requires each vertex to be given a truthful representation of the parameter $t$. In the full version of this paper [6, Appendix A], we show that achieving a runtime $t \in \omega(1) \cap o(n)$ requires that vertices are given some truthful information about $t$ (or $n$).

In this subsection, we describe a verifier for ACYCLIC that only assumes that the space provided to each processor is $O(\log^* n)$. The tradeoff is that our algorithm runs in time which may be linear in $n$ in the worst case.

**Theorem 18.** *There exists a $O(n)$-PLS for* ACYCLIC *which uses labels and space of size $O(\log^* n)$. In each round, the communication per-edge is $O(1)$.*

**Remark 19.** *While verification time in Theorem 18 is $O(n)$ in the worst case, the actual time depends on the labels given to the vertices. In particular, for every acyclic graph $G$ there exists a correct labeling which will be accepted in time $O(\log D)$. Thus there is a tradeoff between the time of the algorithm and the amount of truthful information about $t$ given to the vertices.*

The idea of the algorithm is to simulate the verifier VERIFY (Algorithm 1) without the benefit of truthful information about $t$. As before, the labels designate blocks of length $t$. Within each block, the vertices store shares of the distance of that block to the root, where in this case, the shares consist of a single bit. Since $t$ (the length of the block) is not known to the vertices in advance, they must first compute $t$. However, storing $t$ requires $\log t$ bits, so the computed value of $t$ is stored in shares in sub-blocks of length $\log t$. In order to verify the correctness of the sub-blocks, the vertices must count to $\log t$ using $\log \log t$ bits of memory. This value is again stored in shares in sub-sub-blocks of length $\log \log t$. This process of recursively verifying the lengths of blocks continues until the block length is constant. Thus $\log^* n$ levels of recursion suffice.

Formally, in our recursive scheme, ***recursive acyclicity labels*** closely resemble those in Subsect. 4.2. For each vertex $v$ and each level $i = 1, 2, \ldots, k = \log^* n$, we have an associated block label $b_i(v)$ and block color $c_i(v)$. We refer to the labels associated to each $i$ as a ***level***, denoted $L_i$. The top level $L_1$ additionally contains orientation labels, $a(v)$ and distance labels $d(v)$ for each vertex.

Each level $i$ has an associated length, denoted by $t_i$. We emphasize that the $t_i$ are not initially known to the vertices at the beginning of an execution. The semantics and correctness of the block labels $b_i$ and block colors $c_i$ are precisely the same as those described in Subsect. 4.2, where blocks at level $i$ have length $t_i$. As before, the distance labels $d(v)$ encode (a share of) the purported distance of the $L_1$ block containing $v$ to the root.

**Definition 20.** *Suppose $\ell$ is a family of recursive acyclicity labels for a graph $G = (V, E)$. We say that a family $\ell$ of recursive acyclicity labels is **correct** if the $L_1$ labels are correct as in Definition 14, and for $i \geq 2$ the block labels in $b_i$ and block colors $c_i$ are correct as in Definition 14 with $t_i = \lfloor \log t_{i-1} \rfloor$.*

**Remark 21.** *For simplicity of presentation, we assume that for all $i \geq 2$ that $t_i$ divides $t_{i-1}$. Thus, each block in $L_{i-1}$ contains an integral number of sub-blocks. The general case can be obtained by allowing "overlap" of the last sub-block of $B$ in level $i$ with the first sub-block of $B'$ in $i$ where $B$ is the parent block of $B'$.*

Analogously to Proposition 16, we obtain the following result.

**Proposition 22.** *Let $G = (V, E)$ be a graph. Then $G$ is acyclic if and only if it admits a correct family $\mathcal{C}$ of recursive acyclicity labels.*

It is clear that recursive acyclicity labels are of length $O(\log^* n)$. Indeed, each of the labels in the $\log^* n$ recursive levels has length $O(1)$.

**Lemma 23.** *Let $G = (V, E)$ be a graph, and $\mathcal{C}$ a family of recursive acyclicity labels on $G$. Suppose that for some $i$, the labels in $L_{i+1}$ are correct. Then there exists a verifier $\mathbf{v}_i$ for the labels in $L_i$ with run-time $O(2^{t_{i+1}})$, constant communication per round, and constant space.*

**Algorithm 2.** RVERIFY$(i, L_i)$

```
1: verify a is correct
2: verify properties 1 and 3 of correctness
   of b_i correctness of c_i
3: if i = log* n then
4:     verify correctness of b_i and c_i
5:     return
6: end if
7: TCOUNT_{i+1} ← 0
8: COUNT(TCOUNT_{i+1}, 1, i)
9: SEND(TCOUNT_{i+1}, REC, i + 1)
10: assert: REC_{i+1} = TCOUNT_{i+1}
11: if i = 1 then
12:     ADD(d(v), TCOUNT_2, DCOUNT, 1)
13:     SEND(DCOUNT, DCOUNT, 1)
14:     assert: DCOUNT = d(v)
15: end if
```

We describe a verifier RVERIFY (Algorithm 2) for $L_i$ assuming $L_{i+1}$ is correct. Suppose $B$ is a block in level $i$, and $B_1, B_2, \ldots, B_s$ its sub-blocks for $s = t_i/t_{i+1}$, with $B_j$ the parent of $B_{j+1}$. By assumption, the block labels for the $B_j$ are correct. The head $v_0$ of $B$ verifies that it is also the head of $B_1$, and sends a

token $T_{\text{count}}$ to all of its children. The vertices in $B$ bounce $T_{\text{count}}$ to the tail, which then bounces $T_{\text{count}}$ back up to $v_0$. Meanwhile, the vertices of each $B_j$ hold shares of a counter $\text{TCOUNT}_j$, which computes $t_i$ by incrementing itself until $T_{\text{count}}$ returns to the head. If the counter $\text{TCOUNT}_j$ ever exceeds $2^{t_{i+1}}$ (i.e., if the bit held by the tail of $B_j$ is ever incremented twice), then the vertices in $B_j$ will halt and reject the label. It is clear that this step of the verification will always halt in time $O(2^{t_{i+1}})$. After counting, the blocks in $L_{i+1}$ verify that they agree on $\text{TCOUNT}_j$. Further, tails of $B_j$ verify that their share of TCOUNT is 1, implying that $2^{t_{i-1}-1} < t_i \leq 2^{t_{i-1}}$.

There is a slight complication in the verification algorithm described above that arises when a block $B$ terminates prematurely in a leaf (a vertex of degree 1) which is not a tail. In correct block labels, if $v_0$ is the head of overlapping ***complete*** blocks (i.e., all have tails at distance $t_i$ from the head) then $v_0$ should receive $T_{\text{count}}$ from all of its children at the same time, $2t_i$. However, if some block containing $v_0$ is ***incomplete*** (terminates prematurely with a leaf) then $v_0$ may receive messages from its children in different rounds. To avoid this problem, leaves which are not labeled TAIL respond with a token $T_{\text{leaf}}$ to their parent upon receiving $T_{\text{count}}$. The parent then knows not to expect a $T_{\text{count}}$ from this child. Similarly, if an internal vertex receives $T_{\text{leaf}}$ from all of its children (perhaps in different rounds), it sends $T_{\text{leaf}}$ to its parent. Then vertices check that they receive $T_{\text{count}}$ from all children at the same time, except those which have sent $T_{\text{leaf}}$ if a previous round.

Finally, if $i = 1$, the vertices must additionally verify the correctness of the distance labels $d(v)$. Suppose $B = (v_0, \ldots, v_{t-1})$ and $B' = (w_0, \ldots, w_{t-1})$ are blocks with $B$ the parent of $B'$. The tail $v_{t-1}$ sends $b(v_{t-1})$, $c(v_{t-1})$, and $d(v_{t-1})$ to its children, and sends the token $T_{\text{start}}$ to its parent, $v_{t-2}$. The vertices continue to echo any messages received from their parents to their children, and if a vertex $v$ receives $T_{\text{start}}$ from its children, it additionally sends $b(v)$, $c(v)$, and $d(v)$ to its children. When $w_{t-1}$ (the tail of $B'$) receives $d(v_{t-1})$, it saves this value and sends $T_{\text{stop}}$ to its parent. When a vertex $w$ receives $T_{\text{stop}}$, it saves the value $d(v)$ in the message it received from its parent such that $c(v) \neq d(v)$, and echos $T_{\text{stop}}$ to its parent. After $2t$ rounds, the procedure terminates, and every $w_i$ holds $d(v_i)$. In a further $3t$ rounds, $B'$ distributively increments the $d(v_i)$, and verify that the incremented $d(v_i)$ are equal to $d(w_i)$, thus ensuring the distance labels are correct.

*Proof (of Lemma 23).* We prove that RVERIFY$(i, L_i)$ (Algorithm 2) is a verifier for $L_i$ whenever $L_{i+1}$ is a correct. As in the proof of Lemma 17, we focus on verifying properties 2 and 4 in the correctness of $b_i$. Properties 1 and 3 of the correctness of $b_i$, as well as the correctness of $c_i$ can be trivially verified in a single communication round with constant communication. Let $v_0$ be a root in $L_i$. By induction, every vertex at distance $\tau$ from $v_0$ receives $T_{\text{count}}$ at time $\tau$. Thus, property 4 of the correctness of $b_i$ is satisfied if and only if no vertex fails in a call to COUNT$(\text{TCOUNT}_{i+1}, 1, i)$, which occurs if and only if each $2^{t_{i+1}-1} < \text{TCOUNT}_{i+1} \leq 2^{t_{i+1}}$ (line 20 of COUNT ensures the first inequality, while the check in lines 11–13 of INCREMENT ensure the second inequality). Property 2 in

the correctness of $b_i$ holds if and only if all vertices accept the assertion at line 10 of RVERIFY$(i, L_i)$.

The proof that $d$ is correct when $i = 1$ if and only if no vertex rejects in lines 11–15 in RVERIFY$(i, L_i)$ is analogous to the argument in Lemma 17. Finally, it is clear that the per-round communication is constant, as is the space requirement (assuming that only levels $L_i$ and $L_{i+1}$ are stored). As for the run-time, notice that COUNT$(\text{CTR}, m, i)$ always terminates in time at most $2^{mt_{i+1}}$ by the verification at lines 11–13 of INCREMENT. Further, if no vertex fails during the call to count COUNT, then ADD and SEND will similarly halt after $2^{t_{i+1}} \leq t_i$ rounds.

*Proof (of Theorem 18).* By Proposition 22, it suffices to prove the existence of a verifier $\mathbf{v}$ of recursive acyclicity labels with the claimed communication, space, and time. We induct on $k-i$ (where $k = \log^* n$) that the correctness of $L_i$ can be verified in the desired run-time, using constant communication and space. When $i = k$, the correctness of labels is a local property (independent of the size of the network). Thus, each vertex $v$ can verify the correctness of $L_k$ by analyzing the state of $L_k$ labels in $N(v, O(1))$, which can be accomplished in constant time, space, and communication. Now suppose the correctness of $L_{i+1}$ can be verified in time $O(t)$ using constant communication and space. By Lemma 23, RVERIFY$(i, L_i)$ (Algorithm 2) is a verifier for $L_i$. Further, RVERIFY$(i, L_i)$ runs in time $O(t_i) \leq O(\log(t_1))$, uses constant communication, and space. Theorem 18 the follows by running RVERIFY$(k, L_k)$, followed by RVERIFY$(k-1, L_{k-1})$ and so on, up to RVERIFY$(1, L_1)$. The run-time is $O(t_k + t_{k-1} + \cdots + t_1) \leq O(t_1)$.

**Remark 24.** *We can modify the recursive scheme described here to use only finitely many levels of recursion, but with the tradeoff of using more memory per-vertex. In particular, if only the labels of $L_1$ are given, but each vertex has access to a counter with $\log t$ bits of memory, we recover precisely the scheme of Subsect. 4.2 in the case where $t = \Omega(\log n)$. If we give labels in $L_1$ and $L_2$, and each vertex has a counter with $\log\log t$ bits of memory, then the scheme will still be correct. However, we get a greater degradation of run-time due to round-off errors in $\log\log t$. Specifically, if we have $m - 1 < \log\log t \leq m$, then we obtain*

$$2^{2^{m-1}} < t \leq \left(2^{2^{m-1}}\right)^2.$$

*Thus, even if $\log\log t$ is given truthfully as the size of the counter, the run-time of* RVERIFY *may be quadratic in $t$ if the $L_1$ labels are improperly formed. Finally, given labels $L_1$, $L_2$, and $L_3$, and a counters of size $\log^{(3)} t$, the run-time may vary exponentially from $\log n$. Thus, our worst-case run-time is already only $O(n)$. The fully recursive scheme thus achieves the same worst-case run-time with $\log^* n$ memory per vertex.*

## References


1. Afek, Y., Kutten, S., Yung, M.: The local detection paradigm and its application to self-stabilization. Theor. Comput. Sci. **186**(1–2), 199–229 (1997)
2. Awerbuch, B., Ostrovsky, R.: Memory-efficient and self-stabilizing network reset (extended abstract). In: Proceedings of 13th Annual ACM Symposium on Principles of Distributed Computing, PODC 1994, pp. 254–263. ACM, New York (1994)
3. Awerbuch, B., Patt-Shamir, B., Varghese, G.: Self-stabilization by local checking and correction. In: 32nd Symposium on Foundations of Computer Science (FOCS), pp. 268–277. IEEE (1991)
4. Baruch, M., Fraigniaud, P., Patt-Shamir, B.: Randomized proof-labeling schemes. In: Proceedings of 2015 ACM Symposium on Principles of Distributed Computing, PODC, pp. 315–324 (2015)
5. Baruch, M., Ostrovsky, R., Rosenbaum, W.: Brief announcement: space-time tradeoffs for distributed verification. In: Proceedings of 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, pp. 357–359. ACM, New York (2016)
6. Baruch, M., Ostrovsky, R., Rosenbaum, W.: Space-time tradeoffs for distributed verification. CoRR, arXiv:1605.06814 (2016)
7. Blin, L., Fraigniaud, P., Patt-Shamir, B.: On proof-labeling schemes versus silent self-stabilizing algorithms. In: Felber, P., Garg, V. (eds.) SSS 2014. LNCS, vol. 8756, pp. 18–32. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11764-5_2
8. Das Sarma, A., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., Wattenhofer, R.: Distributed verification and hardness of distributed approximation. SIAM J. Comput. **41**(5), 1235–1265 (2012)
9. Feuilloley, L., Fraigniaud, P.: Survey of distributed decision. Bull. EATCS, **119** (2016)
10. Foerster, K.-T., Luedi, T., Seidel, J., Wattenhofer, R.: Local checkability, no strings attached. In: Proceedings of 17th International Conference on Distributed Computing and Networking, ICDCN 2016, pp. 21:1–21:10. ACM, New York (2016)
11. Fraigniaud, P., Korman, A., Peleg, D.: Towards a complexity theory for local distributed computing. J. ACM **60**(5), 35 (2013)
12. Fraigniaud, P., Rajsbaum, S., Travers, C.: Locality and checkability in wait-free computing. Distrib. Comput. **26**(4), 223–242 (2013)
13. Göös, M., Suomela, J.: Locally checkable proofs. In: 30th ACM Symposium on Principles of Distributed Computing (PODC), pp. 159–168 (2011)
14. Itkis, G., Levin, L.: Fast and lean self-stabilizing asynchronous protocols. In: Proceedings of 35th Annual Symposium on Foundations of Computer Science, SFCS 1994, pp. 226–239. IEEE Computer Society, Washington, DC (1994)
15. Korman, A., Kutten, S.: Distributed verification of minimum spanning trees. Distrib. Comput. **20**, 253–266 (2007)
16. Korman, A., Kutten, S., Masuzawa, T.: Fast and compact self stabilizing verification, computation, and fault detection of an MST. In: 30th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 311–320 (2011)
17. Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. Distrib. Comput. **22**(4), 215–233 (2010)
18. Schmid, S., Suomela, J.: Exploiting locality in distributed SDN control. In: Proceedings of 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, pp. 121–126. ACM, New York (2013)

# Approximate Proof-Labeling Schemes

Keren Censor-Hillel[1(✉)], Ami Paz[1], and Mor Perry[2]

[1] Department of Computer Science, Technion, Haifa, Israel
{ckeren,amipaz}@cs.technion.ac.il
[2] Department of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel
mor@eng.tau.ac.il

**Abstract.** We study a new model of verification of boolean predicates over distributed networks. Given a network configuration, the proof-labeling scheme (PLS) model defines a distributed proof in the form of a label that is given to each node, and all nodes locally verify that the network configuration satisfies the desired boolean predicate by exchanging labels with their neighbors. The *proof size* of the scheme is defined to be the maximum size of a label.

In this work, we extend this model by defining the *approximate proof-labeling scheme* (APLS) model. In this new model, the predicates for verification are of the form $\psi \leq \varphi$, where $\psi, \varphi : \mathcal{F} \to \mathbb{N}$ for a family of configurations $\mathcal{F}$. Informally, the predicates considered in this model are a comparison between two values of the configuration. As in the PLS model, nodes exchange labels in order to locally verify the predicate, and all must accept if the network satisfies the predicate. The soundness condition is relaxed with an approximation ration $\alpha$, so that only if $\psi > \alpha\varphi$ some node must reject.

We show that in the APLS model, the proof size can be much smaller than the proof size of the same predicate in the PLS model. Moreover, we prove that there is a tradeoff between the approximation ratio and the proof size.

**Keywords:** Distributed graph algorithms · Distributed verification
Approximation algorithms · Primal-dual algorithms

## 1 Introduction

### 1.1 Context and Objective

Verification of a given property in decentralized systems finds applications in various domains, such as, checking the result obtained from the execution of a distributed program [5,20], establishing lower bounds on the time required for distributed approximation [11], estimating the complexity of logic required for distributed run-time verification [21], general distributed complexity theory [19], and the construction of self stabilizing algorithms [8,26].

K. Censor-Hillel and A. Paz—Supported by ISF individual research grant 1696/14.
M. Perry—Partially supported by Apple Graduate Fellowship.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 71–89, 2017.
https://doi.org/10.1007/978-3-319-72050-0_5

In the distributed setting, a network configuration $G_s$ is represented by an underlying graph and a state assignment. The nodes of the underlying graph represent processors and the edges represent communication links between pairs of processors. The state assignment is the state of each node, which can contain a unique identifier, edge weights, a specified subset of incident edges, an output of a distributed algorithm and more. In order to verify that a network configuration has a specified property, nodes exchange messages along the edges and output either TRUE or FALSE depending on whether the local configuration is consistent with a legal state of the network. The distributed verification process is correct if all nodes return TRUE on legal configurations, and on every illegal configuration at least one node returns FALSE. Some properties are local by nature and easy to verify, for example, whether a specified subset of edges is a matching in the graph. However, many other properties cannot be verified in less than diameter time, even if message size and local computational power are unbounded, for example, whether a specified matching is of maximum cardinality.

In order to cope with strong time lower bounds, Korman et al. [27] have introduced the *proof-labeling schemes* (PLSs) computational model, where nodes are given auxiliary global information in the form of *labels*. A proof-labeling scheme for a predicate $\mathcal{P}$ consists of a *prover* and a *verifier*. For every legal state of the network, the prover assigns a *label* to every node. The verifier is a distributed algorithm, in which nodes exchange labels with their immediate neighbors and then output TRUE or FALSE at each node, as a function of the state and label of the node and the labels it receives from its neighbors. A PLS satisfies *completeness* if for every legal configuration, with the labels assigned by the prover, all nodes output TRUE, and it satisfies *soundness* if for every illegal configuration and every label assignment, some node outputs FALSE.

When designing a PLS, we wish to minimize the maximum size of a label, which is called the *proof size*. It is known that, for every sequentially decidable graph property, there exists a PLS with proof size $O(m \log n)$ where $n$ is the number of nodes and $m$ is the number of edges in the network [6,22,27]. For some properties, lower bounds on the proof size have been proven in this model, for example $\Omega(\log n)$ for verification of a spanning-tree [27] and bi-connectivity [6], $\Omega(n^2/\log n)$ for verifying that the graph is not 3-colorable [22], and $\Omega(\log^2 n)$ for verification of a minimum-weight spanning-tree [25], assuming that the maximal edge-weight $W$ satisfies $\log n < W \leq n^c$ for some constant $c$.

As in the computational framework, variations of the model may allow us to break known lower bounds. It has been suggested to use super-constant number of rounds in verification [7,26]. In the former, a linear reduction of proof size is proven for acyclicity and the universal scheme. In the latter, they present a scheme for minimum-weight spanning-tree with $O(\log n)$ proof size and $O(\log^2 n)$ rounds. In [6] it was suggested to distinguish between labels and communication in the verification process, and to use randomization in order to reduce the communication complexity of verification. They show an exponential reduction in the communication complexity of every scheme at the cost of increasing the proof size by a factor of the maximum degree.

**Table 1.** APLS for $(D \leq k)$ on general graphs—upper and lower bounds on proof size.

| Approximation ratio | Upper bound | | Lower bound | |
|---|---|---|---|---|
| Exact | $O(n \log n)$ | (Section 3) | $\Omega(n/k)$ | (Theorem 1) |
| $3/2 - \epsilon$ | | | $\Omega\left(n/\log^2 n\right)$ | (Theorem 3) |
| 3/2 | $O\left(\sqrt{n}\log^2 n\right)$ | (Theorem 2) | | |
| 2 | $O(\log n)$ | (Theorem 4) | | |

Yet, some properties are still harder. In Sect. 3 we show that any PLS for $D \leq k$ must have labels of $\Omega(n)$ bits, where $D$ is the diameter of the graph and $k \in \mathbb{N}$ is a constant. A natural way to circumvent this lower bound is through approximation, e.g., by defining a 2-approximation for the problem by the predicate $D \leq 2k$, and hoping for smaller proof size. However, this approach is bound to fail: any PLS for $D \leq 2k$ is also a PLS for $D \leq k'$, for $k' = 2k$, so the same lower bound holds for this definition of approximation.

Inspired by the above example, we present and investigate a new concept of *approximate* proof-labeling schemes (APLSs for short) for optimization problems. Let $\psi, \varphi : \mathcal{F} \to \mathbb{N}$ be two functions from a family of configurations to the natural numbers. Assume that we are interested in verifying for every $G_s \in \mathcal{F}$ whether $\psi(G_s) \leq \varphi(G_s)$, and let $\alpha > 1$ be the approximation ratio. If $\psi(G_s) \leq \varphi(G_s)$ then there is an assignment of labels such that all nodes output TRUE, and if $\psi(G_s) > \alpha\varphi(G_s)$ then for every label assignment at least one node outputs FALSE. If $\varphi(G_s) < \psi(G_s) \leq \alpha\varphi(G_s)$, we do not have any promise. Put differently, we are promised that if all nodes output TRUE, then $\psi(G_s) \leq \alpha\varphi(G_s)$, i.e., the approximation holds. This concept indeed allows us to find schemes with shorter labels: we show a 2-APLS for $D \leq k$ with proof size of only $O(\log n)$ bits, and a 3/2-APLS for $D \leq k$ with proof size of $O(\sqrt{n}\log^2 n)$ bits.

### 1.2 Our Contribution

In this paper we introduce and formalize the concept of *approximate* proof-labeling schemes. We study the complexity of verification of two fundamental problems in this model: diameter and maximum weight matching. We start by considering the verification of a specified upper bound $k$ on the network diameter $D$ (see summary of results in Table 1), and show that for every $k = k(n)$, the proof size of any PLS for $D \leq k$ is $\Omega(n/k)$. In the APLS model, as outlined above, we present a 3/2-APLS for $D \leq k$ with $O(\sqrt{n}\log^2 n)$ proof size, and prove that we cannot obtain a better approximation ratio with the same asymptotic proof size. Specifically, we prove that for every $k$ there exists an $\epsilon \in \Theta(1/k)$ such that the proof size of any $(3/2 - \epsilon)$-APLS for $D \leq k$ is $\Omega(n/\log^2 n)$. Then, we turn to show that if we increase the approximation ratio we can construct an even more efficient scheme. In particular, we show a simple 2-APLS for $D \leq k$ with proof size $O(\log n)$. To our knowledge, the problem of verifying an upper bound on the diameter in general graphs has not been studied before in the context of PLSs.

**Table 2.** APLS for $(w(M) \geq w(\text{MWM}))$—upper and lower bounds on proof size.

| Approximation ratio | Graph family | Upper bound | | Lower bound |
|---|---|---|---|---|
| Exact | Paths | $O(\log n + \log W)$ | [27] | |
| Exact | Bipartite | $O(\log W)$ | [22] | |
| 2 | Trees | $O(\log n + \log W)$ | [27] | |
| 2 | Any graph | $O(\log W)$ | (Theorem 6) | |
| Any | Any graph | | | $\Omega(1)$ (Sect. 4) |

The second property we consider is verifying that a specified matching $M$ have the maximum possible weight (see summary of results in Table 2). For this property we are interested in bounding from below the weight of the matching w.r.t. the weight of the maximum matching $w(\text{MWM})$. We present a 2-APLS for $w(M) \geq w(\text{MWM})$ with $O(\log W)$ proof size, where $W$ is the maximum edge-weight in the network. This improves upon a previous result presented in [27], with $O(\log n + \log W)$ proof size for a 2-approximation of the maximum weight matching on trees. We note that the notion of approximation in [27] is different from our definition: they argue that there exists a subset of 2-approximated configurations that the scheme verifies, but do not promise that any configuration with an optimal matching is verified successfully.

We use various techniques to obtain our results. The lower bounds for proof complexity are achieved using reductions for nondeterministic communication complexity [22], a lower bound graph presented in [23] and a recent constructions of [1]. The APLSs' design is based on approximation algorithms for the diameter problem [2], and on complementary slackness conditions for primal-dual problems.

### 1.3 Related Work

Approximation algorithms were studied extensively in both sequential and distributed computing. In the sequential model, unless P = NP, there are no polynomial-time algorithms for NP-hard problems, and thus efficient approximation algorithms for the related optimization problems are widely studied [31]. Moreover, even for problems for which polynomial time algorithms exist, there is sometimes a need for faster algorithms that give an approximate solution.

One example is the problem of determining the diameter of a graph. While the problem is solvable in polynomial time, faster approximation algorithms are studied. A trivial 2-approximation algorithm in unweighted graphs goes through building a single BFS tree in $O(n+m)$ time, and measuring its depth. An $\tilde{O}(m\sqrt{n}+n^2)$-time 3/2-approximation algorithm for the diameter was presented in [2], and was later improved in [30] to $\tilde{O}(m\sqrt{n})$ time algorithms using randomization. A deterministic improvement to [2] was presented in [9]. Distributed algorithms for computing the diameter were presented in [23,29], and both also provide approximation algorithms for the problem. Lower bounds on computing and approximating the diameter in the CONGEST model were presented in [1,24].

Distributed decision and verification schemes deal with verifying that a given instance satisfies some given boolean predicate. These were formalized in various models to suit its myriad applications, which include proof-labeling schemes (PLSs) [27], locally checkable proofs (LCP) [22], and several complexity classes [19]. The complexity classes presented in the latter include LD—local decision—which includes all properties that can be decided using a constant number of rounds and no additional information, and NLD—non-deterministic local decision—which includes all properties that can be decided in a constant number of rounds with additional information in the form of a certificate given to each node. While NLD and PLS are closely related, they differ in that NLD certificates are independent of node identifiers. Since PLS labels may depend on node identifiers, there is a PLS for every sequentially decidable property on ID based networks, while not all sequentially decidable properties are in NLD. For more details, we refer the reader to a survey of this field of research [12].

The concept of PLS was introduced by Korman et al. in [27]. Among other results, they show a $\Theta(\log n)$ bound on the proof size of the diameter of trees, and the same bound also for the proof size of a lower bound on the diameter in general graphs. In addition, they present two $O(\log n + \log W)$ schemes to verify a maximum weight matching: one on paths, and the other is a 2-approximation of maximum weight matching on trees.

Proof labeling schemes where nodes may communicate to a constant distance that is greater than 1 were studied in [22]. For the maximum cardinality matching problem, they show that the proof size on the family of bipartite graphs is $\Theta(1)$, and on the family of cycle graphs is $\Theta(\log n)$. For maximum weight matching, they present a scheme for the family of bipartite graphs, with $O(\log W)$ proof size, using techniques similar to the ones we use. Moreover, [22] was the first to use nondeterministic communication complexity lower bounds in order to achieve lower bounds on the verification complexity of a PLS.

Schemes with super-constant verification time were presented in [26]. Verification processes in which the global result is not restricted to be the conjunction of local outputs had been studied in [3,4]. The role of unique node identifiers in local decision and verification was extensively studied in [16–18]. The use of randomization in verification process in order to reduce communication was presented in [6]. Proof-labeling schemes in directed networks were studied in [14], where both one-way and two-way communication over directed edges had been considered. Verification schemes for dynamic networks, where edges may appear or disappear after label assignment and before verification, were studied in [15]. Finally, a hierarchy of local decision as an interaction between a prover and a disprover was presented in [13].

## 2 Model and Definitions

### 2.1 Computational Framework

A network is modeled by a connected, undirected, simple graph $G = (V, E)$, with $|V| = n$ nodes and $|E| = m$ edges. Each node represents a processor, and each

edge represents a communication link. We do not assume the a processor initially knows to which other processors it is connected, but only that its communication links are enumerated by *port numbers*. A *configuration* $G_s$ is graph $G = (V, E)$ along with a state assignment function $s : V \to S$, where $S$ is called the *state space*. The state $s(v)$ of a node $v$ includes all local input to $v$. In particular, the state includes port numbers of adjacent edges, the node's identity (if the network is not anonymous) or other data, e.g., the result of an algorithm. We sometimes consider *weighted networks*, in which the graph is accompanied with an edge weight function $w : V \to \{1, \ldots, W\}$, in which case the state of a node includes the weights of its adjacent edges.[1]

In this work, we always assume non-anonymous networks, i.e., every node $v$ is provided with a unique identity $\mathrm{ID}(v)$, which is part of the state of $v$.

### 2.2 Proof-Labeling Schemes

Given a family $\mathcal{F}$ of network configurations and a boolean predicate $\mathcal{P}$ over $\mathcal{F}$, a *proof-labeling scheme* (*PLS*) for $(\mathcal{F}, \mathcal{P})$ is a mechanism for deciding $\mathcal{P}(G_s)$ for every $G_s \in \mathcal{F}$. A PLS consists of two components: a *prover* $\mathbf{p}$, and a *verifier* $\mathbf{v}$. Given any legal configuration $G_s \in \mathcal{F}$ (i.e., a configuration satisfying $\mathcal{P}$), the prover assigns a bit string $\ell(v)$ to every node $v$, called the *label* of $v$. The verifier is a local distributed algorithm running concurrently at every node. At each node $v$, it takes as input the state $s(v)$ of $v$, its label $\ell(v)$ and the labels of all its neighbors, i.e., the list $(\ell(v_1) \ldots \ell(v_d))$, where $d$ is the degree of $v$, and $v_i$ is the neighbor of $v$ reachable from port number $i$. The outputs of the verifier at each node is a boolean value. If the outputs are TRUE at all nodes, $\mathbf{v}$ is said to *accept* the configuration, and otherwise (i.e., $\mathbf{v}$ outputs FALSE in at least one node) $\mathbf{v}$ is said to *reject* the configuration. For correctness, a PLS $(\mathbf{p}, \mathbf{v})$ for $(\mathcal{F}, \mathcal{P})$ must satisfy the following requirements, for every $G_s \in \mathcal{F}$:

- If $\mathcal{P}(G_s) =$ TRUE then, using the labels assigned by $\mathbf{p}$, the verifier $\mathbf{v}$ accepts $G_s$.
- If $\mathcal{P}(G_s) =$ FALSE then, for every label assignment, the verifier $\mathbf{v}$ rejects $G_s$.

The *proof size* of a PLS $(\mathbf{p}, \mathbf{v})$ is the maximum length of a label assigned by the prover $\mathbf{p}$ on a legal configuration $G_s \in \mathcal{F}$.

### 2.3 The New Model: Approximate Proof-Labeling Schemes

In this paper we focus on predicates that represent minimization or maximization problems. Formally, we are given two functions $\psi, \varphi : \mathcal{F} \to \mathbb{N}$, and we are interested in the predicate $\psi(G_s) \leq \varphi(G_s)$. Note that $\psi$ or $\varphi$ may be constant, e.g., in verifying an upper bound on the diameter of the graph, one can be interested in verifying $D(G_s) \leq k$. In some cases, classic verification might be too expansive, as proven in Sect. 3, and so we extend the definition of PLSs to

[1] Recall that $W$ is the maximum weight of an edge in the graph. If $W = 1$, we interpret $O(\log W)$ as $O(1)$.

*approximate proof-labeling schemes* (APLSs). We relax the requirements of a PLS so that a configuration for which the inequality $\psi(G_s) \leq \varphi(G_s)$ holds is guaranteed to be accepted by the scheme, while a configuration for which $\psi(G_s)$ much larger than $\varphi(G_s)$ is guaranteed to be rejected. Formally, for $\alpha \geq 1$, an $\alpha$-APLS $(\mathbf{p}, \mathbf{v})$ for $(\mathcal{F}, (\psi \leq \varphi))$ must satisfy the following requirements, for every $G_s \in \mathcal{F}$:

- If $\psi(G_s) \leq \varphi(G_s)$ then, using the labels assigned by $\mathbf{p}$, the verifier $\mathbf{v}$ accepts $G_s$.
- If $\psi(G_s) > \alpha\varphi(G_s)$ then, for every label assignment, the verifier $\mathbf{v}$ rejects $G_s$.

The *proof size* of an APLS is defined similarly to that of a PLS. Our definitions naturally extend to predicates of the form $\psi \geq \varphi$, $\psi < \varphi$ and $\psi > \varphi$.

Finally, we note that although the definition of an APLS might seem to resemble definitions from the field of property testing, they are inherently different. Our measure for how close a graph is to satisfy a property is entirely algebraic, and has nothing to do with changing the graph by adding or removing edges. Moreover, all schemes presented in this paper are deterministic.

### 2.4 Problem Definitions

*Diameter.* Given a configuration $G_s$ with an underlying graph $G = (V, E)$ and an edge weight function $w$, for every two nodes $u, v \in V$ denote by $\text{dist}(u, v)$ the length of the shortest (unweighted) path between $u$ and $v$ in $G_s$, and by $\text{dist}_w(u, v)$ the minimum weight of a path between $u$ and $v$ in $G_s$. The *unweighted diameter* of $G_s$, denoted by $D(G_s)$, is defined as $\max\{\text{dist}(u, v) \mid u, v \in V\}$. Similarly, The *weighted diameter* of $G_s$, denoted by $D_w(G_s)$, is defined as $\max\{\text{dist}_w(u, v) \mid u, v \in V\}$.

The first set of problems we consider in this work are problems of bounding the weighted and unweighted diameters from above.

**Definition 1.** *Let $\mathcal{F}$ be the family of all weighted connected undirected configurations and let $G_s \in \mathcal{F}$. For every integer $k = k(n)$, we define the problems $(\mathcal{F}, (D \leq k))$ and $(\mathcal{F}, (D_w \leq k))$.*

A *breadth-first search* (BFS) tree in a weighted or unweighted graph $G_s$ from a root $r \in V$ is a tree consisting of a shortest (unweighted) path from $r$ to every node in $V$. If the graph is weighted, we are also interested in a *shortest weighted distance tree* consisting of a shortest weighted path from a root node $r$ to every node in $V$. Throughout the paper, we use known schemes for verification of a BFS tree and a shortest weighted distance tree [27]. They prove that for the verification of these trees it is enough to give every node the identity of the root and the distance from the root. Therefore, proof size is $O(\log n)$ for a BFS tree and $O(\log n + \log W)$ for a shortest weighted distance tree.

*Matchings.* Given a configuration $G_s$ with an underlying graph $G = (V, E)$, an edge weight function $w$, and an edge subset $M \subset E$, $M$ is a *matching* in $G$ if no two edges in $M$ share a node. The weight of a matching $M$, denoted by $w(M)$, is the sum of weights of all edges in $M$. We say that a matching $M$ is a maximum weight matching (MWM) if $w(M) \geq w(M')$ for every matching $M'$ in $G$.

Another problem we consider, of a different flavor, is to verify that a specified matching is a maximum weight matching.

**Definition 2.** *Let $\mathcal{F}_M$ be the family of all weighted connected undirected configurations with a specified matching $M$. Let $G_s \in \mathcal{F}$ and let* MWM *be a maximum weight matching in $G_s$. We define the problem* $(\mathcal{F}_M, (w(M) \geq w(\text{MWM})))$.

Note that although $w(M) > w(\text{MWM})$ is not possible (since $M$ is promised to be a matching), the problem is defined to follow the structure of APLSs.

### 2.5 Two-Party Communication Complexity

Given two vectors $x, y \in \{0,1\}^s$, we say the vectors are not disjoint, and write $\text{DISJ}(x, y) = \text{FALSE}$, if there exists an index $i \in [s]$ such that $x_i = y_i = 1$. Otherwise, the vectors are disjoint, and $\text{DISJ}(x, y) = \text{TRUE}$. In the Set-Disjointness two-party communication problem, two players denoted Alice and Bob are given two vectors, $x, y \in \{0,1\}^s$ respectively, and they need to decide whether $\text{DISJ}(x, y) = \text{TRUE}$ or $\text{DISJ}(x, y) = \text{FALSE}$. (See [28] for complete definitions and discussion.)

Given their inputs, the players communicate by a deterministic *protocol*, and eventually output $\text{DISJ}(x, y) = \text{TRUE}$ or $\text{DISJ}(x, y) = \text{FALSE}$. A well known result in communication complexity asserts that in any protocol, Alice and Bob must exchange $\Omega(s)$ bits in order to correctly determine the value of $\text{DISJ}(x, y)$.[2]

In the nondeterministic case of the problem, Alice and Bob use auxiliary bit strings, which each of them nondeterministically chooses, and then run a deterministic protocol in order to determine the value of $\text{DISJ}(x, y)$. We are interested in the best assignment of auxiliary strings, i.e. the one that allows the players to minimize the number of bits exchanged. For example, if $\text{DISJ}(x, y) = \text{FALSE}$ and Alice and Bob both use the index $i$ such that $x_i = y_i = 1$ as an auxiliary string, then they only need to exchange $O(\log s)$, to verify they have the same index. On the other hand, a celebrated result [28] asserts that when $\text{DISJ}(x, y) = \text{TRUE}$, Alice and Bob must communicate $\Omega(s)$ even with an optimal assignment of auxiliary strings, i.e. nondeterminism cannot help Alice and Bob in asymptotically minimizing the communication.

## 3 PLS and APLS for Diameter

Verifying that the diameter of the graph is bounded from above by a specified value can be done by a PLS with $O(n \log n)$ proof size (and $O(n(\log n + \log W))$

[2] This lower bound holds also for randomized protocols, which we do not discuss in this work.

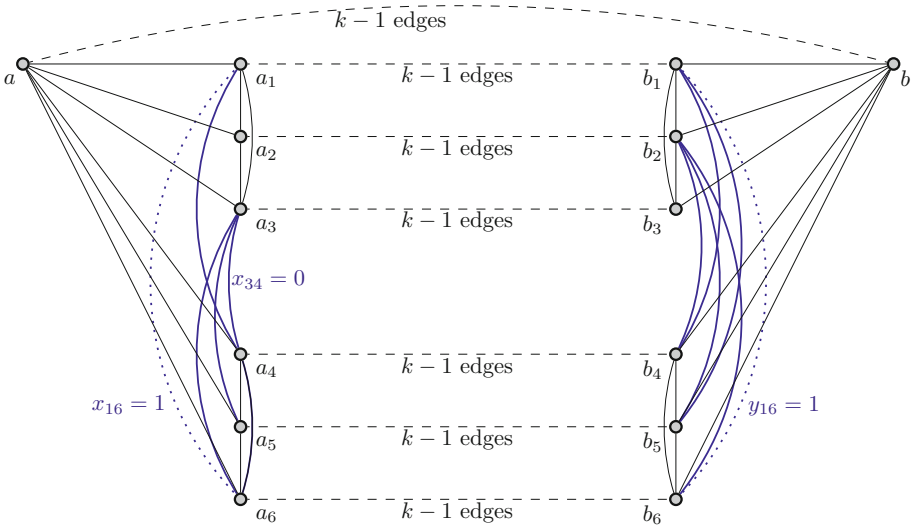



**Fig. 1.** The diameter lower bound construction for $s = 6$. Here, $x = \left(\begin{smallmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{smallmatrix}\right)$ and $y = \left(\begin{smallmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{smallmatrix}\right)$, where the matrix rows are indexed by $\{1, 2, 3\}$ and the columns by $\{4, 5, 6\}$. Since $x_{16} = y_{16} = 1$, the dotted edges are missing and the distance between $a_1$ and $b_6$ is greater than $k$.

for weighted diameter). Simply construct a BFS tree (respectively, a shortest weighted distance tree) from every node, verify it and locally verify at each node that all of its distances are bounded by the specified value. We now show that in the PLS model, for a constant bound $k$, the proof size cannot be improved by more than a $\Theta(\log n)$ factor, i.e., it must have an $\Omega(n)$ proof size. Moreover, for every $k = k(n)$, we show a lower bound for the PLS proof size.

Consider the following graph family $\{G_{x,y}\}$ over $n$ nodes (Fig. 1). Assume $s = \frac{n}{k} - 1$ is an even integer, and let $A_1 = a_1, ..., a_{s/2}$, $A_2 = a_{s/2+1}, ..., a_s$, $B_1 = b_1, ..., b_{s/2}$, and $B_2 = b_{s/2+1}, ..., b_s$ be four cliques, where each $a_i$ is connected to $b_i$ with a path of length $k - 1$, consisting of $a_i$, $b_i$, and $k - 2$ new nodes unique to this path. An additional node $a$ is connected to every $a_i$ by an edge, an additional node $b$ is connected to every $b_i$ by an edge, and there is a $(k - 1)$-node path connecting $a$ and $b$ with another new $k - 2$ nodes. Given an instance $(x, y)$ of the Set-Disjointness problem over $(s/2)^2$ elements, enumerate Alice's input as $x_{ij}$ with $i \in \{1, \ldots, s/2\}$ and $j \in \{s/2 + 1, \ldots, s\}$, and similarly for Bob's input, $y_{ij}$. To complete the construction of $G_{x,y}$, add an edge $(a_i, a_j)$ if and only if $x_{ij} = 0$, and we add an edge $(b_i, b_j)$ if and only if $y_{ij} = 0$.

If $\frac{n}{k} - 1$ is not an even integer, we choose $s$ to be the largest even integer such that $s < \frac{n}{k} - 1$, add nodes to described construction to complement the number of nodes to $n$, and connect all additional nodes to all neighbors of $b$.

**Lemma 1.** *$D(G_{x,y}) \leq k$ if and only if* $\text{DISJ}(x,y) = \text{TRUE}$.

*Proof.* If $\text{DISJ}(x,y) = \text{TRUE}$, then for each $\{i,j\}$, at least one of the edges $(a_i,a_j)$ or $(b_i,b_j)$ exists in $G_{x,y}$. Let $u$ and $v$ be any two nodes in $G_{x,y}$. Suppose that $u$ is on the path $(a_i \rightsquigarrow b_i)$ and $v$ is on the path $(a_j \rightsquigarrow b_j)$, where $i,j \in \{1,\ldots,s/2\}$. If $i=j$, clearly, $\text{dist}(u,v) \leq k-1$. Otherwise, by assumption, either the cycle $(a \to a_i \rightsquigarrow b_i \to b_j \rightsquigarrow a_j \to a)$ or the cycle $(b \to b_j \rightsquigarrow a_j \to a_i \rightsquigarrow b_i \to b)$ exists and its length is $2k+1$. Hence, every two nodes in the cycle are at distance at most $k$ from each other, and $\text{dist}(u,v) \leq k$. Suppose now that either $u$ or $v$ is on the path $(a \rightsquigarrow b)$ and the other node is on the path $(a_i \rightsquigarrow b_i)$, $i \in \{1,\ldots,s\}$. The length of the cycle $(a \to a_i \rightsquigarrow b_i \to b \rightsquigarrow a)$ is $2k$, and since $u$ and $v$ are on this cycle, $\text{dist}(u,v) \leq k$. Finally, if both $u$ and $v$ are on the path $(a \rightsquigarrow b)$, clearly, $\text{dist}(u,v) \leq k-1$, and we conclude that $D(G_{x,y}) \leq k$.

If $\text{DISJ}(x,y) = \text{FALSE}$, then there exist $i \in \{1,\ldots,s/2\}$ and $j \in \{s/2+1,\ldots,s\}$ such that $x_{ij} = y_{ij} = 1$, and by the construction of $G_{x,y}$, both edges $(a_i,a_j)$ and $(b_i,b_j)$ are absent. Every path from $a_i$ to $b_j$ must go through some $(a' \rightsquigarrow b')$ path of length $k-1$, and if $\text{dist}(a_i,b_j) \leq k$ then the shortest path connecting $a_i$ and $b_j$ can only contain one more edge. However, since the edges $(a_i,a_j)$ and $(b_i,b_j)$ are both absent in $G_{x,y}$, no such path exists, so $\text{dist}(a_i,b_j) > k$, which implies that $D(G_{x,y}) > k$. □

**Theorem 1.** *For every $k$, the proof size of any PLS for $(\mathcal{F},(D \leq k))$ is $\Omega(n/k)$.*

*Proof.* Consider any PLS for $(\mathcal{F},(D \leq k))$, and construct a nondeterministic protocol for $\text{DISJ}(x,y)$ as follows. Alice and Bob simulate the verification of $D(G_{x,y}) \leq k$ using the PLS, such that Alice simulates the nodes in $A = A_1 \cup A_2 \cup a$, and Bob simulates the rest of the nodes, denoted by $B$. Each of the players nondeterministically chooses the labels of its nodes as his auxiliary bit-string. Alice and Bob then exchange the labels corresponding to the nodes touching the cut, and simulate the verification process in all nodes. Then, they compute $a$ and $b$, the conjunction of the returned values of $A$ and $B$ respectively. Finally, Alice sends $a$ to Bob, Bob sends $b$ to Alice, and they both output the conjunction $a \wedge b$ as the solution for $\text{DISJ}(x,y)$.

If $\text{DISJ}(x,y) = \text{TRUE}$ then $D(G_{x,y}) \leq k$, there is an assignment of labels to the nodes such that all nodes output TRUE, and if both players choose these labels as their bit-strings then they both output $\text{DISJ}(x,y) = \text{TRUE}$. On the other hand, if $\text{DISJ}(x,y) = \text{FALSE}$ then $D(G_{x,y}) > k$, for every assignment of labels to the nodes at least one node outputs FALSE, and Alice and Bob output $\text{DISJ}(x,y) = \text{FALSE}$ in all executions.

Thus, the simulation we presented is a nondeterministic protocol for deciding $\text{DISJ}(x,y)$. We know that in any nondeterministic protocol for Set-Disjointness $(s/2)^2$ elements, Alice and Bob must exchange $\Omega((s/2)^2)$ bits. The number of edges in the cut of $G_{x,y}$ induced by the partition of the nodes between Alice and Bob in the simulation is $s+1$. Therefore, the proof size of any PLS for $(\mathcal{F},(D \leq k))$ is $\Omega(s) \in \Omega(n/k)$. □

We now show that in the APLS model there are schemes with much smaller proof size. We start with a 3/2-APLS and construct a scheme that is based on

the randomized algorithm for a 3/2-approximation of the diameter presented in [30]. We use the following two lemmas.

**Lemma 2.** *Let $G = (V, E)$ be a graph, let $S, N \subseteq V$ be two sets of nodes, and consider a node $w \in V$. Assume that $N$ is the set of $z$ nodes closest to $w$ for some parameter $z$, $w$ is the farthest node from the set $S$, and $N \cap S$ is non-empty. Then, the largest depth $D'$ of a BFS tree rooted at a node in $R = N \cup S \cup \{w\}$ satisfies $\frac{2}{3}D \leq D' \leq D$.*

**Lemma 3.** *Let $G = (V, E)$ be a graph and $z \in \mathbb{N}$ a parameter. For each $v \in V$, let $N_z(v)$ be the set of $z$ nodes closest to $v$. Then, there exists a hitting set for $\{N_z(v) \mid v \in V\}$, of size $O(n \log n/z)$.*

Lemma 2 corresponds to an adapted version of Lemma 4 of [30], and Lemma 3 is a corollary of Theorem 2.7 of [2]. We obtain the following result.

**Theorem 2.** *There exists a 3/2-APLS for $(\mathcal{F}, (D \leq k))$ with proof size $O(\sqrt{n} \log^2 n)$.*

*Proof.* Our scheme is based on Lemma 2: it consists of a node $w$, sets $N$ and $S$ and all the BFS trees rooted at $R = N \cup S \cup \{w\}$. In addition, there is a node $w'$ that is used to verify that the largest depth of a BFS tree rooted in $R$ is as claimed, and a BFS tree rooted at $w'$. The main task in our scheme is to verify the BFS trees described above, and that the diameter estimation, i.e., the maximum depth of the trees, is at most $k$. Since a BFS tree verification is known from previous work, the challenges in the scheme construction is to verify locally that $w$ is indeed the farthest node from the set $S$, that $N$ is the neighborhood of $w$, and that the estimation is indeed the maximum depth of a tree.

Formally, let $G_s \in \mathcal{F}$ be a configuration with the underlying graph $G = (V, E)$ and $D(G_s) \leq k$. For every $v \in V$, denote by $N_{\sqrt{n}}(v)$ the $\sqrt{n}$ nodes closest to $v$ (break ties according to IDs), and let $S \subset V$ be a set of $O(\sqrt{n} \log n)$ nodes such that $S$ hits $\left\{N_{\sqrt{n}}(v) \mid v \in V\right\}$, whose existence follows from Lemma 3.

Let $h(v) = \min\{\text{dist}(v, u) \mid u \in S\}$, the distance of $v$ from the set $S$, and let $w$ be the farthest node from $S$, i.e., $h(w) \geq h(v)$ for every $v \in V$. Let $q(w)$ be the largest distance from $w$ to any node in $N_{\sqrt{n}}(w)$. Let $R = S \cup \{w\} \cup N_{\sqrt{n}}(w)$ be a set of $|R| = O(\sqrt{n} \log n)$ nodes, and consider the set $R_{BFS}$ of BFS trees rooted at nodes in $R$. Let $d_{\max}$ be the maximum depth of a tree in $R_{BFS}$ and let $w'$ be a node at distance $d_{\max}$ from one of the roots.

The label assigned to a node $v \in V$ is

$$\ell(v) = (\ell_{\text{BFSs:}S}(v), \ell_{\text{BFSs:}N}(v), \ell_{\text{BFS:}w}(v), \ell_{\text{BFS:}w'}(v), \ell_{hw}(v), \ell_{qw}(v), \ell_{\text{maxdist}}(v))$$

where $\ell_{\text{BFSs:}S}(v)$ is a set of $O(\sqrt{n} \log n)$ pairs $\{(\text{ID}(u), \text{dist}(v, u)) \mid u \in S\}$; $\ell_{\text{BFSs:}N}(v)$ is a set of $\sqrt{n}$ pairs $\left\{(\text{ID}(u), \text{dist}(v, u)) \mid u \in N_{\sqrt{n}}(w)\right\}$; $\ell_{\text{BFS:}w}(v) = (\text{ID}(w), \text{dist}(v, w))$; and $\ell_{\text{BFS:}w'}(v) = (\text{ID}(w'), \text{dist}(v, w'))$. Every pair mentioned above is the label needed in order to verify the correct structure of the corresponding BFS tree. In order to verify that $w$ is indeed the farthest node from $S$, every node is given the distance of $w$ from $S$, $\ell_{hw}(v) = h(w)$; To verify

the consistency of $N_{\sqrt{n}}(w)$, every node is given the radius of this neighborhood $\ell_{qw}(v) = q(w)$; and $\ell_{\text{maxdist}}(v) = d_{\max}$ is given in order to verify the existence and maximality of the estimation $d_{\max}$.

In the verification process, a node $v$ exchanges labels with all its neighbors, and verifies the following conditions:

1. Consistency of global parameters: For every neighbor $v'$ of $v$, it holds that $\ell_{hw}(v') = \ell_{hw}(v)$, $\ell_{qw}(v') = \ell_{qw}(v)$, and $\ell_{\text{maxdist}}(v') = \ell_{\text{maxdist}}(v)$.
2. All distances are bounded by $d_{\max}$ and $k$: For every pair $(\text{ID}, d)$ in $\ell_{\text{BFSs:}S}(v) \cup \ell_{\text{BFSs:}N}(v) \cup \{\ell_{\text{BFS:}w}(v)\} \cup \{\ell_{\text{BFS:}w'}(v)\}$, it holds that $0 \leq d \leq \ell_{\text{maxdist}}(v) \leq k$.
3. Existence of a BFS tree of depth $d_{\max}$: If $\ell_{\text{BFS:}w'}(v) = (\text{ID}(v), 0)$ then there exists a pair $(\text{ID}, d) \in \ell_{\text{BFSs:}S}(v) \cup \ell_{\text{BFSs:}N}(v) \cup \{\ell_{\text{BFS:}w}(v)\}$ such that $d = \ell_{\text{maxdist}}(v)$.
4. Only one pair for each node in $S$ and in $N_{\sqrt{n}}(w)$: For every two pairs $(\text{ID}, d), (\text{ID}', d') \in \ell_{\text{BFSs:}X}(v)$, for $X \in \{S, N\}$, if $d \neq d'$ then $\text{ID} \neq \text{ID}'$.
5. BFS structures: For every neighbor $v'$ of $v$ and $X \in \{S, N\}$, the following holds. There exists a pair $(\text{ID}, d) \in \ell_{\text{BFSs:}X}(v)$, for some $d$ if and only if there exists a pair $(\text{ID}, d') \in \ell_{\text{BFSs:}X}(v')$ with the same ID and $d' \in \{d-1, d, d+1\}$. For $x \in \{w, w'\}$, $\ell_{\text{BFS:}x}(v) = (\text{ID}, d)$ for some $d$ if and only if $\ell_{\text{BFS:}x}(v') = (\text{ID}, d')$ for $d' \in \{d-1, d, d+1\}$.
6. Existence of roots: For every $X \in \{S, N\}$ and pair $(\text{ID}, d) \in \ell_{\text{BFSs:}X}(v)$, if $d > 0$ then there exists a neighbor $v'$ of $v$ with $(\text{ID}, d-1) \in \ell_{\text{BFSs:}X}(v')$. For $x \in \{w, w'\}$, if $\ell_{\text{BFS:}x}(v) = (\text{ID}, d)$ and $d > 0$ then there exists a neighbor $v'$ of $v$ with $\ell_{\text{BFS:}x}(v') = (\text{ID}, d-1)$.
7. Unique roots: For every pair $(\text{ID}, d)$ in $\ell_{\text{BFSs:}S}(v) \cup \ell_{\text{BFSs:}N}(v) \cup \{\ell_{\text{BFS:}w}(v)\} \cup \{\ell_{\text{BFS:}w'}(v)\}$, if $d = 0$ then $\text{ID} = \text{ID}(v)$.
8. Non-empty intersection of $S$ and $N_{\sqrt{n}}(w)$: There exists a pair $(\text{ID}, d) \in \ell_{\text{BFSs:}S}(v) \cap \ell_{\text{BFSs:}N}(v)$.
9. Maximality and correctness of $h(w)$: There exists a pair $(\text{ID}, d) \in \ell_{\text{BFSs:}S}(v)$ such that $d \leq \ell_{hw}(v)$, and if $\ell_{\text{BFS:}w}(v) = (\text{ID}(v), 0)$ then there exists no pair $(\text{ID}, d) \in \ell_{\text{BFSs:}S}(v)$ such that $d < \ell_{hw}(v)$.
10. The neighborhood of $w$: Let $\ell_{\text{BFS:}w}(v) = (\text{ID}, d)$. If $d < \ell_{qw}(v)$ then there exists a pair $(\text{ID}(v), 0) \in \ell_{\text{BFSs:}N}(v)$, and if $d > \ell_{qw}(v)$ then there exists no pair $(\text{ID}(v), 0) \in \ell_{\text{BFSs:}N}(v)$.

The completeness of this 3/2-APLS follows from the fact that if $D(G_s) \leq k$ then the maximum depth of any BFS tree in $G_s$ is at most $k$.

For the soundness, consider a configuration $G_s \in \mathcal{F}$ with the underlying graph $G = (V, E)$ and label assignment $\ell$, and assume that all nodes output TRUE. By (1), all nodes have the same values $\ell_{hw}$, $\ell_{qw}$ and $\ell_{\text{maxdist}}$. By (4), (5), (6) and (7) for every node $v \in V$ and every pair $(\text{ID}, d) \in \ell_{\text{BFSs:}S}(v) \cup \ell_{\text{BFSs:}N}(v) \cup \{\ell_{\text{BFS:}w}(v)\} \cup \{\ell_{\text{BFS:}w'}(v)\}$, there exists a node $u$ such that $\text{ID} = \text{ID}(u)$ and it holds that $d = \text{dist}(v, u)$.

Let $\overline{S}(v)$ be the collection of IDs in $\ell_{\text{BFSs:}S}(v)$, let $\overline{N}(v)$ be the collection of IDs in $\ell_{\text{BFSs:}N}(v)$, let $\overline{w}(v)$ be the ID in $\ell_{\text{BFS:}w}(v)$ and let $\overline{w'}(v)$ be the ID

in $\ell_{\mathrm{BFS}:w'}(v)$. By (5), for every two nodes $v$ and $u$ it holds that $\overline{S}(v) = \overline{S}(u)$, $\overline{N}(v) = \overline{N}(u)$, $\overline{w}(v) = \overline{w}(u)$ and $\overline{w'}(v) = \overline{w'}(u)$. We denote these values by $\overline{S}$, $\overline{N}$, $\overline{w}$ and $\overline{w'}$ respectively. By (10), $\overline{N}$ is the set of closest nodes to $\overline{w}$; by (9), $\overline{w}$ is the farthest node from the set $\overline{S}$; and by (8), there exists some node in the intersection of $\overline{N}$ and $\overline{S}$. By (3), the collection of pairs $\ell_{\mathrm{BFS}:w'}(v)$ of all nodes $v \in V$ indicates a BFS rooted at $\overline{w_0}$ with distance $\ell_{\mathrm{maxdist}}$ to one of the nodes in $\overline{S} \cup \overline{N} \cup \{\overline{w}\}$, and by (2) we know that this is the largest distance from any node to one of the nodes in $\overline{S} \cup \overline{N} \cup \{\overline{w}\}$ and this distance is at most $k$.

Overall, we have a collection of BFS trees with depth at most $\ell_{\mathrm{maxdist}} \leq k$. Therefore, all conditions of Lemma 2 are satisfied, and we have $(2/3)D(G_s) \leq \ell_{\mathrm{maxdist}}$. Hence, $D(G_s) \leq (3/2)k$ as desired.

The proof size follows from Lemma 3, which implies that there exists a set $S$ of size $O(\sqrt{n}\log n)$ that is a hitting set for $\left\{N_{\sqrt{n}}(v) \mid v \in V\right\}$. In particular, the intersection $N_{\sqrt{n}}(w) \cap S$, where $w$ is the farthest node from $S$, is not empty. Therefore, the label consists of $O(\sqrt{n}\log n)$ sub-labels of size $O(\log n)$ each. □

The following result shows that with the proof size we obtain for 3/2-APLS we cannot have a better approximation ratio that is correct for all possible bounds $k$. To get a better approximation ratio, one needs to use labels that are almost as large as the labels used for exact PLS.

Let $x$ and $y$ be two $s$-bit strings, $s \in \Omega(n/\log n)$. Our lower bound follows the recent construction of Abboud et al. [1].[3]

**Lemma 4** [1]. *Given two strings $x, y \in \{0,1\}^s$, there exists a graph $G_{x,y} = (V, E)$ and a partition of $V$ into $V_A$ and $V_B$ such that:*

1. *The number of nodes in $G_{x,y}$ is $n \in \Theta(s \log s)$.*
2. *All the edges depending on $x$ are between nodes in $V_A$.*
3. *All the edges depending on $y$ are between nodes in $V_B$.*
4. *The number of edges between nodes in $V_A$ and $V_B$ is in $\Theta(\log s)$.*
5. *If $\mathrm{DISJ}(x, y) = \textsc{true}$ then $D(G_{x,y}) \leq k$, and otherwise $D(G_{x,y}) > 3k/2 - 9$.*

From this construction we derive the following lower bound.

**Theorem 3.** *For every $k$, there exists an $\epsilon \in \Theta(1/k)$ such that the proof size of any $(3/2 - \epsilon)$-APLS for $(\mathcal{F}, (D \leq k))$ is $\Omega(n/\log^2 n)$.*

*Proof.* Consider a $(3/2 - 9/k)$-APLS for $(\mathcal{F}, (D \leq k))$, and an instance $(x, y)$ of the DISJ problem over $s$ bits. Construct the graph $G_{x,y}$ as in Lemma 4, with the same partition to $V_A$ and $V_B$. Alice and Bob nondeterministically choose the labels for the nodes of $V_A$ and $V_B$, simulate the verification algorithm together, and then compute $a$ and $b$, the conjunction of the returned values of $V_A$ and $V_B$. Finally, Alice sends $a$ to Bob, Bob sends $b$ to Alice, and they both output the conjunction $a \wedge b$ as the solution for $\mathrm{DISJ}(x, y)$.

By Lemma 4, if $\mathrm{DISJ}(x, y) = \textsc{true}$ then $D \leq k$, all nodes must accept and Alice and Bob return TRUE. On the other hand, If $\mathrm{DISJ}(x, y) = \textsc{false}$ then

[3] See Chap. 2.2 of [1]. We use $P = \lfloor (k-2)/4 \rfloor$.

$D > (3/2 - 9/k)k$, at least one node rejects, and Alice and Bob return FALSE. Thus, Alice and Bob correctly solve the Set-Disjointness problem over $s$ elements.

Note that $\log n = \Theta(\log s)$. Alice and Bob must communicate $\Omega(s) = \Omega(n/\log n)$ bits, and there are $O(\log n)$ nodes touching the cut, so the proof size is $\Omega(n/\log^2 n)$. □

To further study the tradeoff between the approximation ratio and the proof size, we now prove that if we increase the approximation ratio we can construct an even more efficient scheme.

**Theorem 4.** *There exists a 2-APLS for* $(\mathcal{F}, (D_w \leq k))$ *with proof size* $O(\log n + \log W)$.

*Proof.* Let $G_s \in \mathcal{F}$ such that $D_w(G_s) \leq k$, and let $r \in V$ be some node. The label assigned to every node $v \in V$ is $\ell(v) = (\ell_{\text{dist}}(v), \ell_{\text{root}}(v))$, where $\ell_{\text{dist}}(v) = \text{dist}_w(r, v)$ and $\ell_{\text{root}}(v) = \text{ID}(r)$. To verify that $D_w(G_s) \leq k$, a node $v$ exchanges labels with all its neighbors, and verifies the following conditions:

1. For every neighbor $u$ of $v$, it holds that $\ell_{\text{root}}(u) = \ell_{\text{root}}(v)$.
2. $0 \leq \ell_{\text{dist}}(v) \leq k$.
3. If $\ell_{\text{dist}}(v) > 0$ then $v$ has at least one neighbor $u$ with $\ell_{\text{dist}}(u) = \ell_{\text{dist}}(v) - w(u, v)$.
4. If $\ell_{\text{dist}}(v) = 0$ then $\ell_{\text{root}}(v) = \text{ID}(v)$.

The completeness of this 2-APLS is clear: If $D_w(G_s) \leq k$ and labels are assigned as described above, all nodes output TRUE.

For the soundness, consider a configuration $G_s$ with label assignment $\ell$, such that all nodes output TRUE. For a node $v$ in the graph, follow the path from $v$ constructed by repeatedly going from a node $v'$ to its neighbor $u$ with $\ell_{\text{dist}}(u) = \ell_{\text{dist}}(v') - w(u, v')$, whose existence is guaranteed by Condition (3). By conditions (2) and (3), this path must end after traversing a weight of at most $k$, at a node $r$ with $\ell_{\text{dist}}(r) = 0$, and this node is unique by Conditions (1) and (4). As this claim can be applied to each node in the graph, every two nodes in the graph are connected to each other by a path through $r$, of weighted distance at most $2k$, and $D_w(G_s) \leq 2k$ as desired. □

The following corollary directly follows Theorem 4 for the unweighted case.

**Corollary 1.** *There exists a 2-APLS for* $(\mathcal{F}, (D \leq k))$ *with proof size* $O(\log n)$.

## 4 Maximum Weight Matching

Given a configuration $G_s \in \mathcal{F}_M$ with the underlying graph $G = (V, E)$, an edge weight function $w$, and a specified matching $M \subset E$, we wish to verify $(\mathcal{F}_M, (w(M) \geq w(\text{MWM})))$. Göös and Suomela [22] present a PLS for this problem in bipartite graphs, using a linear programming (LP) formulation. Here, we extend their technique to present a 2-APLS for $(\mathcal{F}_M, (w(M) \geq w(\text{MWM})))$ on general graphs.

Our 2-APLS is simple: the label of a matched node is the weight of its matched edge, and the label of an unmatched node is 0. The verification process, and the proof that this is indeed a 2-APLS are slightly more involved, and use a relaxation of the complementary slackness conditions of a relaxation of a linear-programming formulation for the problem.

Consider the next integral-LP formulation of the MWM problem (cf. [10, Chap. 5]):

$$\begin{array}{ll} \text{Maximize} & \sum_{e\in E} w(e)x_e \\ \text{Subject to} & \sum_{\{e|v\in e\}} x_e \leq 1, \forall v \in V \\ & x_e \in \{0,1\}, \qquad \forall e \in E, \end{array}$$

and the LP obtained by relaxing the integrality condition into:

$$x_e \geq 0, \quad \forall e \in E.$$

The dual linear-program of the relaxed problem is

$$\begin{array}{ll} \text{Minimize} & \sum_{v\in V} y_v \\ \text{Subject to} & y_u + y_v \geq w(e), \forall e = (u,v) \in E. \end{array}$$

Given a pair consisting of a primal and a dual feasible solutions, their optimality can be verified by checking several conditions derived from the LP, conditions that are known as the *complementary slackness conditions*. For the aforementioned LP, the conditions are:

$$\begin{array}{l} x_e > 0 \implies y_u + y_v = w(e), \quad e = (u,v) \in E; \text{ and} \\ y_v > 0 \implies \sum_{\{e|v\in e\}} x_e = 1, \; v \in V. \end{array}$$

If $G$ is bipartite, then any pair of feasible optimal solutions satisfy the complementary slackness conditions, a fact that lies at the heart of the PLS presented by Göös and Suomela [22].

For general graphs, the same method fails miserably. The inherent obstacle that this approach faces is the integrality gap of the LP formulation: a fractional solution to the problem may be twice as large as the maximum integral solution. While there are LP formulations of the problem with an integrality gap of 1, it is not clear how to translate them into a PLS, since the number of dual variables in these LPs is substantially larger.

However, we observe that a relaxed version of these conditions is enough to prove that a primal solution is an approximation of the MWM.

**Theorem 5 (See** [31, Sect. 15.1]**).** *If $x$ and $y$ are feasible primal and dual solutions in a graph $G$ satisfying*

$$\begin{array}{ll} x_e > 0 \implies & w(e) \leq y_u + y_v \leq 2w(e), \; e = (u,v) \in E; \text{ and} \\ y_v > 0 \implies & \sum_{\{e|v\in e\}} x_e = 1, \qquad\qquad v \in V, \end{array}$$

*then $x$ is a 2-approximation of the MWM in $G$.*

Unlike the case of bipartite graphs, here the opposite implication does not hold: not every pair of 2-approximate solutions fulfill the conditions. Thus, given a matching represented by a vector $x$, we explicitly build a dual solution $y$ such that $x$ and $y$ satisfy above conditions. This dual solution $y$ will serve as a 2-APLS for $(\mathcal{F}_M, (w(M) \geq w(\text{MWM})))$ in a general graph.

**Theorem 6.** *There exists a 2-APLS for* $(\mathcal{F}_M, (w(M) \geq w(\text{MWM})))$ *with proof size* $O(\log W)$.

*Proof.* Let $G$ be a weighted graph with weights in $\{1, \ldots, W\}$ and $M$ a maximum weight matching in $G$. Let $(x_e)_{e \in E}$ be the indicator vector of $M$. Define the values of the dual variables $(y_v)_{v \in V}$ by $y_v = w(e)$ if there exist an edge $e \in M$ such that $v \in e$, and $y_v = 0$ otherwise. The label of a node $v$ is set to be $y_v$.

To verify $(\mathcal{F}_M, (w(M) \geq w(\text{MWM})))$, a node $v$ exchanges labels with its neighbors and check the next feasibility condition:

– For each neighbor $u$ of $v$, $y_u + y_v \geq w(u, v)$.

We start by showing that if $M$ is indeed a MWM, then the relaxed complementary slackness conditions hold. Let $e = (u, v)$ be an edge satisfying $x_e > 0$, i.e. $e \in M$, then $y_u = y_v = w(e)$ and indeed $w(e) \leq y_u + y_v \leq 2w(e)$. For the second complementary slackness condition, let $v$ be a node with $y_v > 0$, so there is exactly one edge $(u, v) \in M$ with $x_{(u,v)} = 1$, while for every other neighbor $u'$ of $v$, $x_{(u',v)} = 0$, so $\sum_{\{e|v \in e\}} x_e = 1$.

For the feasibility, the input is a feasible matching, so $\sum_{\{e|v \in e\}} x_e \leq 1$ for each node $v$ and $x_e \geq 0$ for each edge $e$, and the primal solution $x$ is feasible. For the dual solution $y$, assume towards contradiction that there is an edge $e = (u, v)$, $e \notin M$, such that $y_u + y_v < w(e)$. Then, the matching obtained by removing any edge in $M$ that touches $u$ or $v$ and adding $e$ to $M$ has a weight $w(M) - (y_u + y_v) + w(e) > w(M)$, which contradicts the maximality of $M$. The case of $e \in M$ was considered in the previous paragraph. Thus, we have a pair of feasible primal and dual solutions satisfying the relaxed slackness conditions, and the solutions are 2-approximations of the optimal solutions.

Finally, consider a configuration $G_s$ with label assignment $(x_e)$, such that all nodes output TRUE. The labels represent a dual solution that satisfies all the relaxed complementary slackness conditions, so by Theorem 5 the solution is a 2-approximation of the MWM. □

We are unaware of any lower bound for the MWM problem in the PLS model, nor in the CONGEST and LOCAL models. We note that for every approximation ratio $\alpha \geq 1$, some communication is needed in any $\alpha$-APLS for $(\mathcal{F}_M, (w(M) \geq w(\text{MWM})))$. This is true since, for every configuration $G_s$ with an empty matching $M = \emptyset$ (not any approximation of MWM), the local view of every node is consistent with some legal configuration with matching $M'$, where $w(M') = w(\text{MWM})$. Let $v$ be a node and let $u_1, \ldots, u_d$ be the neighbors of $v$ where the weight of every edge $(v, u_i)$ is $w_i$. The construction of the legal configuration $G_s^v$ for $v$ is as follows. Add nodes $z_1, \ldots, z_d$ and an edge $e_i = (z_i, u_i)$

of weight $w_i + 1$ for every $1 \leq i \leq d$. Finally, define $M' = \{e_i \mid 1 \leq i \leq d\}$. It is easy to verify that there is no augmenting path for $M'$ in this configuration, i.e., $w(M') = w(\text{MWM})$. However, the local view of $v$ in $G_s$ and in $G_s^v$ is the same. Therefore, without communication, $v$ must output TRUE. Since the same holds for every node, we conclude that some communication is necessary, regardless of the desired approximation ratio.

## 5 Discussion

This paper presents the new model of approximate proof-labeling schemes. We illustrate the power of the APLS model with the $D \leq k$ predicate. We prove a tight lower bound (up to a logarithmic factor) in the PLS model, and present two, more efficient, APLSs for this predicate. The two APLSs show a non-trivial tradeoff between the approximation ratio and the proof size.

We also present a 2-APLS for the predicate $w(M) \geq w(\text{MWM})$ on general graphs, a problem for which it is unknown if a non-trivial PLS exists. Presenting an efficient PLS for this problem, showing that a PLS with small proof size does not exist, or presenting an APLS with different approximation ratio or different proof size are interesting questions left open.

It would be interesting to study the APLS model on other graph predicates. For example, the chromatic number $\chi(G)$ of a graph $G$ is the minimal number of colors in a proper node coloring of $G$. A PLS for $\chi \leq k$ with proof size $O(\log k)$ exists, where the proof is a proper coloring of the graph. However, it was proven in [22] that any PLS for $\chi > 3$ must have $\tilde{\Omega}(n^2)$ proof size. Hence, also for this problem, the APLS model may allow a more efficient verification.

Finally, the idea of approximation in verification we present in this paper can be extended to other decision and verification schemes, such as the complexity classes LD and NLD, generating a different classification of problems. For example, our 2-APLS for $w(M) \geq w(\text{MWM})$ on general graphs can also be used for 2-approximate NLD, under the relevant definitions, since the labels can be locally computed by the nodes.

**Acknowledgment.** We thank Gilad Kutiel, Seffi Naor and Dror Rawitz for discussions of the primal-dual method, and the anonymous reviewers of SIROCCO 2017 for valuable comments.

## References

1. Abboud, A., Censor-Hillel, K., Khoury, S.: Near-linear lower bounds for distributed distance computations, even in sparse networks. In: Gavoille, C., Ilcinkas, D. (eds.) DISC 2016. LNCS, vol. 9888, pp. 29–42. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53426-7_3
2. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J. Comput. **28**(4), 1167–1181 (1999)

3. Arfaoui, H., Fraigniaud, P., Ilcinkas, D., Mathieu, F.: Distributedly testing cycle-freeness. In: Kratsch, D., Todinca, I. (eds.) WG 2014. LNCS, vol. 8747, pp. 15–28. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12340-0_2
4. Arfaoui, H., Fraigniaud, P., Pelc, A.: Local decision and verification with bounded-size outputs. In: Higashino, T., Katayama, Y., Masuzawa, T., Potop-Butucaru, M., Yamashita, M. (eds.) SSS 2013. LNCS, vol. 8255, pp. 133–147. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03089-0_10
5. Awerbuch, B., Patt-Shamir, B., Varghese, G.: Self-stabilization by local checking and correction. In: FOCS, pp. 268–277. IEEE (1991)
6. Baruch, M., Fraigniaud, P., Patt-Shamir, B.: Randomized proof-labeling schemes. In: PODC, pp. 315–324 (2015)
7. Baruch, M., Ostrovsky, R., Rosenbaum, W.: Space-time tradeoffs for distributed verification. CoRR, arXiv:1605.06814 (2016)
8. Blin, L., Fraigniaud, P., Patt-Shamir, B.: On proof-labeling schemes versus silent self-stabilizing algorithms. In: Felber, P., Garg, V. (eds.) SSS 2014. LNCS, vol. 8756, pp. 18–32. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11764-5_2
9. Chechik, S., Larkin, D.H., Roditty, L., Schoenebeck, G., Tarjan, R.E., Williams, V.V.: Better approximation algorithms for the graph diameter. In SODA, pp. 1041–1052 (2014)
10. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. Wiley, New York (1998)
11. Das Sarma, A., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., Wattenhofer, R.: Distributed verification and hardness of distributed approximation. SIAM J. Comput. **41**(5), 1235–1265 (2012)
12. Feuilloley, L., Fraigniaud, P.: Survey of distributed decision. Bull. EATCS **119** (2016)
13. Feuilloley, L., Fraigniaud, P., Hirvonen, J.: A hierarchy of local decision. In: ICALP, pp. 118:1–118:15 (2016)
14. Foerster, K.-T., Luedi, T., Seidel, J., Wattenhofer, R.: Local checkability, no strings attached. In: ICDCN, pp. 21:1–21:10. ACM (2016)
15. Foerster, K.-T., Richter, O., Seidel, J., Wattenhofer, R.: Local checkability in dynamic networks. In: ICDCN, pp. 4:1–4:10. ACM (2017)
16. Fraigniaud, P.: Göös, M., Korman, A., Suomela, J.: What can be decided locally without identifiers? In: PODC, pp. 157–165. ACM (2013)
17. Fraigniaud, P., Halldórsson, M.M., Korman, A.: On the impact of identifiers on local decision. In: Baldoni, R., Flocchini, P., Binoy, R. (eds.) OPODIS 2012. LNCS, vol. 7702, pp. 224–238. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35476-2_16
18. Fraigniaud, P., Hirvonen, J., Suomela, J.: Node labels in local decision. In: Scheideler, C. (ed.) Structural Information and Communication Complexity. LNCS, vol. 9439, pp. 31–45. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25258-2_3
19. Fraigniaud, P., Korman, A., Peleg, D.: Towards a complexity theory for local distributed computing. J. ACM **60**(5), 35 (2013)
20. Fraigniaud, P., Rajsbaum, S., Travers, C.: Locality and checkability in wait-free computing. Distrib. Comput. **26**(4), 223–242 (2013)
21. Fraigniaud, P., Rajsbaum, S., Travers, C.: On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 92–107. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_9

22. Göös, M., Suomela, J.: Locally checkable proofs in distributed computing. Theory Comput. **12**(1), 1–33 (2016)
23. Holzer, S., Peleg, D., Roditty, L., Wattenhofer, R.: Distributed 3/2-approximation of the diameter. In: DISC, pp. 562–564 (2014)
24. Holzer, S., Wattenhofer, R.: Optimal distributed all pairs shortest paths and applications. In: PODC, pp. 355–364 (2012)
25. Korman, A., Kutten, S.: Distributed verification of minimum spanning trees. Distrib. Comput. **20**, 253–266 (2007)
26. Korman, A., Kutten, S., Masuzawa, T.: Fast and compact self stabilizing verification, computation, and fault detection of an MST. In: PODC, pp. 311–320 (2011)
27. Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. Distrib. Comput. **22**(4), 215–233 (2010)
28. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, New York (1997)
29. Peleg, D., Roditty, L., Tal, E.: Distributed algorithms for network diameter and girth. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012. LNCS, vol. 7392, pp. 660–672. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31585-5_58
30. Roditty, L., Williams, V.V.: Fast approximation algorithms for the diameter and radius of sparse graphs. In: STOC, pp. 515–524 (2013)
31. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-662-04565-7

# Global Versus Local Computations: Fast Computing with Identifiers

Mikaël Rabie(✉)

LIP, ENS de Lyon, 69007 Lyon, France
mikael.rabie@ens-lyon.org

**Abstract.** This paper studies what can be computed by using probabilistic local interactions with agents with a very restricted power in polylogarithmic parallel time.

It is known that if agents are only finite state (corresponding to the Population Protocol model by Angluin *et al.*), then only semilinear predicates over the global input can be computed. In fact, if the population starts with a unique leader, these predicates can even be computed in a polylogarithmic parallel time.

If identifiers are added (corresponding to the Community Protocol model by Guerraoui and Ruppert), then more global predicates over the input multiset can be computed. Local predicates over the input sorted according to the identifiers can also be computed, as long as the identifiers are ordered. The time of some of those predicates might require exponential parallel time.

In this paper, we consider what can be computed with Community Protocol in a polylogarithmic number of parallel interactions. We introduce the class $CPPL$ corresponding to protocols that use $O(n \log^k n)$, for some $k$, expected interactions to compute their predicates, or equivalently a polylogarithmic number of parallel expected interactions.

We provide some computable protocols, some boundaries of the class, using the fact that the population can compute its size. We also prove two impossibility results providing some arguments showing that local computations are no longer easy: the population does not have the time to compare a linear number of consecutive identifiers. The *Linearly Local* languages, such that the rational language $(ab)^*$, are not computable.

## 1 Introduction

Population Protocols, introduced by Angluin *et al.* in 2004 [3], corresponds to a model of finite states devices with a very restricted memory using pairwise interactions to communicate and compute a global result. Predicates computable by population protocols have been characterized as being precisely the semilinear predicates; i.e. those equivalent to be definable in first-order Presburger arithmetic [1,3]. Semi-linearity was shown to be sufficient, and necessary. Those predicates use the global multiset of the input.

Later works on population protocols have concentrated on characterizing what predicates on the input configurations can be stably computed in different


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 90–105, 2017.
https://doi.org/10.1007/978-3-319-72050-0_6

variants of the models and under various assumptions. Variants of the original model considered so far include restriction to one-way communications [1], restriction to particular interaction graphs [2]. Various kinds of fault tolerance have been studied for population protocols [12], including the search for self-stabilizing solutions [5]. Some works also include the Probabilistic Population Protocol model that makes a random scheduling assumption for interactions [4,13].

Some works extend this model. The edges of the interaction graph may have states that belong to a constant-size set. This model called the *mediated population protocol* is presented in [19]. The addition on Non-Determinism has been studied in [8]. The research of Self-Stabilization (over some fairness assumption) has been explored in [5,7]. An extension with sensors offering a *cover-time* notion was also studied in [6]. A recent study in [18] also focused on finding the median agent in an extension of the model called Arithmetic Population Protocols.

More generally, the population protocol model shares many features with other models already considered in the literature. In particular, models of pairwise interactions have been used to study the propagation of diseases [17], or rumors [11]. In chemistry, the chemical master equation has been justified using (stochastic) pairwise interactions between the finitely many molecules [15,20]. The variations over the LOCAL model [14] can be seen as a restriction over the interactions (using a graph) but with a set of possible improvements in agents' capacities.

Agents have been endowed with even stronger tools in different models. The *passively mobile protocols* introduced by Chatzigiannakis *et al.* [10] constitutes a generalization of the population protocol model where finite state agents are replaced by agents that correspond to arbitrary Turing machines with $O(S(n))$ space per-agent, where $n$ is the number of agents. As agents remain initially anonymous, only functions over the global input can be computed.

The *community protocols* introduced by Guerraoui and Ruppert [16] are closer to the original population protocol model, assuming *a priori* agents with individual very restricted computational capabilities. In this model, agents are no longer anonymous: each agent has a unique identifier and can only remember $O(1)$ other agent identifiers. Guerraoui and Ruppert [16] using results about the so-called storage modification machines [21], proved that such protocols simulate Turing machines: Predicates computed by this model with $n$ agents are precisely the predicates in $NSPACE(n \log n)$. The sorted input symbols according to the identifiers can be analysed locally by the protocols to compute the right output. In [9], the possibility that identifiers are no longer unique is explored through the *homonym population protocols* model.

### Motivation

Angluin *et al.* in [4], prove that any computable predicate by a Population Protocol can be computed in $O(n \log^5 n)$ expected interactions, as long as there is a unique leader at the beginning. This article includes some arguments leading to the idea that there might exist protocols computing a leader election in

$O(n \log n)$ expected interactions. Doty and Soloveichik proved in [13] that there cannot be a protocol computing a leader so fast. They proved that a protocol needs $\Omega(n^2)$ expected interactions to get to a configuration with a single leader, if every agent is a potential candidate at the beginning.

The exact characterization of what can be computed by populations having unique leaders gave the motivation to look to what can be computed in $O(n \log^k n)$ expected interactions (for any $k > 0$), with the Community Protocols model [16]. We consider, as in [4], that each pair of agents (or identifiers) have the same probability to be chosen at each step of a computation. In [4], it is considered that dividing the number of expected interaction by $n$ provides the expected number of parallel interactions.

Community protocols can be seen as interactions controlled by devices in a social group. For example, identifiers can correspond to phone numbers, and the devices can be applications on smartphones. In this vision, it seems intuitive to consider that a group of individuals do not want to stay too long together to compute some global information. Sorting a group of people depending on phone numbers to look for patterns does not seem intuitive, and hence useful.

This paper introduces the class $CPPL$, corresponding to what can be computed with Community Protocols in a polylogarithmic number of expected parallel interactions (which corresponds to a number of expected interactions bounded above by $n \log^k n$ for some $k$), or a polylogarithmic number of epidemics or broadcasts. We introduce some protocols, proving that the size of the population (or some subset) can be computed in some sense to be explained.

We then show the weakness of this model based on the fact that local computation cannot be performed over the whole input. More precisely, we prove that only a polylogarithmic number of agents can find the next or previous identifier to their own. We also introduce the class of linearly local languages, containing the rational language $(ab)^*$, and prove that none of its elements cannot be computed.

We finish with some comparisons with other computational classes. We introduce a class of Turing Machine trying to match the expressive power of $CPPL$. Those machines use a polylogarithmic space of computation, and is able to use the tools we found. This machine has access to global informations of the input, but can focus locally only on a polylogarithmic number of regions of the input.

The paper is organized as follows: Sect. 2 provides the Community Protocol model introduced in [16] and includes some examples. Section 3 provides some elements and results about fast computing with Population Protocols from [4]. Section 4 explains a way to keep the fairness of our protocols and describes a way to compute the size of the population. Section 5 introduces the notion of Linearly Local languages and proves that these languages are not in $CPPL$. Section 6 provides some complexity bounds on the class $CPPL$.

## 2 Model

We present now the model introduced by Guerraoui and Ruppert in [16]: Agents have unique identifiers, and can store a fixed number of them. Agents can com-

pare 2 identifiers. We consider that, unlike in [9], agents cannot know when two identifiers are consecutive.

This model has been proved in [16] to correspond to $NSPACE(n \log n)$, even when we add a fixed number of byzantine agents. We will not consider byzantine agents in this paper.

**Definition 1.** *A* Community Protocol *is given by seven elements* $(U, B, d, \Sigma, \iota, \omega, \delta)$ *where:*

- $U$ *is the infinite ordered set of identifiers.*
- $B$ *is a finite set of basic states.*
- $d \in \mathbb{N}$ *is the number of identifiers that can be remembered by an agent.*
- $\Sigma$ *is the finite set of entry symbols.*
- $\iota$ *is an input function* $\Sigma \to B$.
- $\omega$ *is an output function* $B \to \{True, False\}$.
- $\delta$ *is a transition function* $Q^2 \to Q^2$, *with* $Q = B \times U \times (U \cup \{\_\})^d$.

*The set* $Q = B \times U \times (U \cup \{\_\})^d$ *of possible states each agents can have is such that each agent carries three elements: its identifier, its state, and* $d$ *slots for identifiers.*

*The transition function* $\delta$ *has two restrictions: Agents cannot store identifiers that they never heard about, and the transitions must only depend on relative position of the identifiers in the slots and on the state in* $B$. *More formally, we have:*

1. *if* $\delta(q_1, q_2) = (q_1', q_2')$, *and id appears in* $q_1'$ *or* $q_2'$ *then id must appear in* $q_1$ *or in* $q_2$.
2. *whenever* $\delta(q_1, q_2) = (q_1', q_2')$, *let* $u_1 < u_2 < \cdots < u_k$ *be the distinct identifiers that appear in any of the four states* $q_1, q_2, q_1', q_2'$. *Let* $v_1 < v_2 < \cdots < v_k$ *be distinct identifiers. If* $\rho(q)$ *is the state obtained from* $q$ *by replacing all occurrences of each identifier* $u_i$ *by* $v_i$, *then we require that* $\delta(\rho(q_1), \rho(q_2)) = (\rho(q_1'), \rho(q_2'))$.

We also add the fact that $\delta$ cannot change the identifier of an agent.

As a convention, we will often call an agent of initial identifier $id \in U$ the agent $id$. We will sometimes write $Id_k$ for the $k$th identifier present in the population. An agent with identifier $id$, in state $q$ and with a list of $d$ identifiers $L = id_1, \ldots, id_d$ will be written in what follows $q_{id,id_1,\ldots,id_d}$.

*Example 1 (Leader Election).* It is possible to compute a Leader Election (a protocol where all agents start in state $L$ from which we want to reach a configuration with a single $L$: the leader), where the leader will be the agent with the smallest identifier, with $O(n \log n)$ expected interactions. As a reminder, without identifiers, a protocol needs $\Omega(n^2)$ expected interactions to elect a leader [13].

Agents will store the identifier of their leader. Here is the protocol, using above notations for rules:

- $B = \{L, N\}$.
- $d = 1$.
- $\Sigma = L$, $\iota(L) = L$ and $\omega(L) = \omega(N) = True$.
- $\delta$ is such that the non-trivial rules (i.e. where at least one state changes) are:

$$\begin{array}{rrcll} L_{id_a,\_} & L_{id_b,\_} & \rightarrow & L_{id_a,\_} & N_{id_b,id_a} \text{ with } id_a < id_b \\ L_{id_a,\_} & N_{id_b,id_c} & \rightarrow & L_{id_a,\_} & N_{id_b,id_a} \text{ with } id_a < id_c \\ L_{id_a,\_} & N_{id_b,id_c} & \rightarrow & N_{id_a,id_c} & N_{id_b,id_c} \text{ with } id_c < id_a \\ N_{id_a,id_b} & N_{id_c,id_d} & \rightarrow & N_{id_a,id_b} & N_{id_c,id_b} \text{ with } id_b < id_d \end{array}$$

To determine the speed of this protocol, it suffices to realize that the final leader actually does an epidemic to spread its identifier (epidemic is defined in Definition 4). An epidemic takes $O(n \log n)$ expected interactions. Thus, the leader election can be performed in $O(n \log n)$ expected interactions. The notions of time and computation are defined in what follows.

*Remark 1.* To ensure that at some point, a single leader remains in the population, Gerraoui *et al.* uses the notion of Fairness introduced in the Population Protocols model [3]. As we work here with probabilistic interaction (each pair of agents has the same probability to interact), the fairness notion will not be needed.

**Definition 2.** *An* Input *is a subset of $U \times \Sigma$ such that any element of $U$ (the elements of $U$ being called* Identifiers*) can appear at most once. Inputs will often be seen as words of $\Sigma^*$, as it is possible to sort the input elements according to the identifiers (recall that we consider that $U$ is ordered). An input $u = u_1 \dots u_n$ is such that the agent with the smallest identifier has input $u_1$, the second has input $u_2 \dots$*

*The* Initial State *of an agent assigned with the identifier $id$ and the input $s$ is $(\iota(s), id, \_^d)$, where $\_^d$ states for $d$ repetitions of the empty slot $\_$.*

*A* Configuration *is a subset of $Q$ where two elements cannot have the same first identifier (i.e. two agents must have two distinct identifiers).*

*A* Step *is the transition between two configurations $C \rightarrow C'$, where only two agents' states may change: we apply to the two agents $a_1$ and $a_2$ the rule corresponding to their respective state $q_1$ and $q_2$, i.e. if $\delta(q_1, q_2) = (q_1', q_2')$ (also written by rule $q_1\ q_2 \rightarrow q_1'\ q_2'$), then in $C'$ the respective states of $a_1$ and $a_2$ are $q_1'$ and $q_2'$. All other agents have the same state in $C$ and $C'$.*

*A configuration has an* Output *$y \in \{True, False\}$ if for each state $b \in B$ present in the population, $\omega(b) = y$. A configuration $C$ is said* Output Stable *if it has an output $y$ and if, for any $C'$ reachable from $C$, $C'$ has also the output $y$.*

*An input $w \in \Sigma^*$ has an* Output *$y \in Y$ if from any reachable configuration from the initial configuration, we can reach an output stable configuration of output $y$. It means that from the input, the protocol will reach with probability 1 an output stable configuration, and there is a single output $y$ reachable. The input is* Accepted *if and only if it has output $True$.*

*A protocol* Computes *a set $L$ if, for any input word $w \in \Sigma^*$, the protocol provides an output, and the protocol accepts $w$ if and only if $w \in L$. We then say*

*that $L$ is* Computable. *We will sometimes say that the protocol is* Las Vegas, *as it will always succeed to provide an output with probability 1.*

*A language is* Computed in $f(n)$ Expected Interactions *if, for any input $w$, the expected number of interactions to reach an output stable configuration is bounded above by $f(|w|)$.*

The Community Protocols model has been fully characterized:

**Theorem 1** [16]**.** *The decisions problems computable by community protocols correspond exactly to the class $NSPACE(n \log n)$.*

*The set of languages computable by community protocols is $NSPACE(n \log n)$.*

Let us introduce now the class we will work with in this paper:

**Definition 3.** *We define the class* CPPL *as the sets of languages that can be recognized by a Community Protocol with $O(n \log^k n)$ expected interactions for some $k \in \mathbb{N}$, where each pair of agents has the same probability to interact at each moment.*

*We say that a function $f$ is* n-polylog *if there exists some $k$ such that we have $f(n) \leq n \log^k n$.*

## 3 Fast Computing Known Results

We introduce here some of the elements and results in [4] by Angluin *et al.* These elements are on the Population Protocols model. It corresponds to the case where agents do not have identifiers.

The results are based under the assumption that the population starts with a unique leader. With community protocols, this assumption will no longer be used, we will always consider the leader to be the agent with the smallest identifier (see Example 1).

We introduce the main result and some tools from [4] that will be used in this paper. We first introduce the notion of epidemics, which will be our main tool to perform computations. We will quickly talk about the Phase Clock Protocol that permits to be sure with high probability that an epidemic had the time to happen. We finish with a complexity result.

### 3.1 Epidemics

The epidemic is the most important probabilistic protocol. Its purpose is to spread or gather information. It will permit for example to get an identifier, to check the state of an agent of a given identifier, to check if there exists some agent in a given state...

The important element with this tool is that an epidemic takes $O(n \log n)$ expected interactions. Intuitively, in parallel, at each step, the number of agents aware of the epidemic doubles, using $O(\log n)$ parallel steps to spread.

**Definition 4** [4]. *An* Epidemic Protocol *is a protocol who spreads some information through an epidemic. The purpose is, for a leader, to* Infect *each agent. More formally, if 0 represents the not infected state and 1 the infected one, there is just a non trivial rule:*

$$1\ \ 0 \rightarrow 1\ \ 1$$

*Most of the time, it will be a leader who will start a spreading of some information. The computation will start in the configuration* $10^{n-1}$ *(one agent in state* 1*, the others being in state* 0*), where 1 represents the leader.*

**Proposition 1** [4]. *Let* $T$ *be the expected number of interactions before an epidemic protocol starting with a single infected agent infects all the other ones. For any fixed* $c > 0$*, there exist positive constants* $c_1$ *and* $c_2$ *such that, for sufficiently large* $n$*, with probability at most* $1 - n^{-c}$*:*

$$c_1 n \log n \leq T \leq c_2 n \log n$$

From this theorem, we know that any epidemic protocol will take $\Theta(n \log n)$ expected interactions. If we are (almost) sure that more than $c_2 n \log n$ interactions occurred, we will be (almost) sure that an epidemic has finished.

To be almost sure that at least $c_2 n \log n$ interactions have occurred, [4] introduced the Phase Clock Protocol. The leader runs a clock between 0 and $m$ for some $m > 0$. Each agent tries to store the current time, following some updating rules. Each time the clock loops (i.e. the leader reaches $m$ and resets the clock), the population is almost sure that at least $c_2 n \log n$ interactions have occurred.

**Proposition 2** [4]. *For any fixed* $c, d_1 > 0$*, there exist two constants* $m$ *and* $d_2$ *such that, for all sufficiently large* $n$*, with probability at least* $1 - n^{-c}$ *the phase clock protocol with parameter* $m$*, completes* $n^c$ *rounds, where the minimum number of interactions in any of the* $n^c$ *rounds is at least* $d_1 n \log n$ *and the maximum is at most* $d_2 n \log n$*.*

This result permits to be sure, with high probability, that for $n^c$ rounds, in each round, an epidemic had the time to occur.

### 3.2 Presburger's Arithmetic

The main result from [4] is that, if the population starts with a unique leader, any computable predicate by a population protocol can be computed with $O(n \log^5 n)$ expected interactions.

**Theorem 2** [4]. *For any predicate* $P$ *definable in Presburger's Arithmetic, and for any* $c > 0$*, there is a probabilistic population protocol with a leader to compute* $P$ *without error that converges in* $O(n \log^5 n)$ *interactions with probability at least* $1 - n^{-c}$*.*

As a reminder, those predicates correspond to boolean combinations of:

- *Threshold Predicate*: $\sum a_i x_i \geq b$, with $a_1, \ldots, a_n, b \in \mathbb{Z}^{n+1}$.
- *Modulo Predicate*: $\sum a_i x_i \equiv b[c]$, with $a_1, \ldots, a_n, b, c \in \mathbb{Z}^{n+2}$.

where $x_i$ corresponds to the number of agents with input $i \in \Sigma$. This also corresponds to semilinear sets.

**Corollary 1.** *Any predicate definable in Presburger's Arithmetic is in CPPL.*

*Proof.* We use the two following facts:

- The Leader Election can be performed in $O(n \log n)$ (see Example 1).
- Any predicate definable in Presburger's Arithmetic can be computed in $O(n \log^5 n)$ expected interactions (see Theorem 2), as long as there is a single leader.

Each agent stores the smallest identifier it has heard about in its Leader slot. It links its internal clock to the leader: if it meets an agent storing a smallest identifier, it acts as if its own clock was at 0, and performs the interaction with the other agent accordingly. Hence, each agent will act as in the protocols of [4] as soon as it hears about the right leader's identifier (in [4], agents start their role in the computation as soon as they get instruction from the leader, or from someone who transmits leader's instruction through an epidemic).

## 4 Some Computable Protocols

We are now able to introduce some probabilistic protocols, including a complex one that encodes the size of the population. Let first introduce the following notion:

**Definition 5.** *We will often talk about* Next *and* Previous. *Those are two functions $U \to U$ that provides, to a given identifier, the next one/previous one present in the population. More formally:*

- $Next(id_a) = \min\{id_b : id_b > id_a\}$.
- $Previous(id_a) = \max\{id_b : id_b < id_a\}$.

*By convention, Next of the highest identifier is the smallest, and Previous of the smallest identifier is the highest one. Thus, these two functions are bijective.*

*Sometimes, Next and Previous will be slots in protocols, with the purpose to find the right identifier corresponding to the function. "Finding its Next" means that the agent needs to put the right identifier in its slot Next.*

### 4.1 From Monte Carlo to Las Vegas Protocols

We considered in the previous section Monte Carlo protocols (i.e. protocols having eventually a probability of failure). We accept that the protocols might have some probability of failure, as long as we can minimize it as much as needed (we use the same bound of $1 - n^{-c}$ as in [4]). Those protocols alone do not compulsory compute any set.

We provide in this paper Monte Carlo descriptions of the protocols. We consider that the protocols also run in parallel a Las Vegas protocol providing the right output with probability 1 (the corresponding Las Vegas protocols exist, as a consequence of Theorem 1). The protocol detects, as in [4], when the Las Vegas protocol should have finished to find the output. At this point, each agent switches its output from the Monte Carlo protocol's to the Las Vegas protocol's. With probability at least $1 - n^{-c}$, this will not change the output.

Here is a small result to justify that we can transform our protocols presented in this paper in Las Vegas ones by multiplying the expected number of interactions by $n^3$:

**Proposition 3.** *Let be a population where all agents has found their $Next$ (see Definition 5). There exists a protocol that simulates an epidemic spread from an agent taking $O(n^3)$ expected interactions, with a success of probability 1. In the new protocol, the agent meets all the other ones in the population.*

*Proof.* We suppose that all agents have already found their $Next$, and we suppose all agents know the leader's identifier, being the smallest identifier in the population. The agent needs to meet the leader, then remembers the $Next$ of the leader, meets it, remembers its $Next$... until it finds the agent having as $Next$ the leader's identifier. At this point, the agent has met all agents in the population.

Each step takes $\frac{n(n-1)}{2}$ expected interactions, and we have $n$ steps. Hence, this protocol takes $O(n^3)$ expected interactions to derandomize the epidemic from the initial agent.

Finding each $Next$ needs at most $O(n^2 \log n)$ expected interactions (which corresponds to the number of interactions expected before every possible interaction has occurred at least once). Detecting when an agent found a new $Next$ is easy: the corresponding agent goes to find the leader to give the information. This latter then resets its computation, spreading the information as in the previous proof. With probability 1, at some point, all agents will have found the right $Next$ and the leader will then reset for the last time the computation.

We will also use some protocols of [4]. Even though some parts use only epidemics, others are trying to detect when something has finally occurred (for example, detect when some state no longer appears in the population). When our Las Vegas protocol will run this detection, it will iterate the epidemic part until it detects the desired fact. In [4], those elements are proved to happen with high probability in a single epidemic. Hence, our expectation will not grow here.

We can prove that this protocol takes at most $O(n^2 \log n + n^2 \log n + n^3)$ (i.e. $O(n^3)$) expected interactions to reset for the last time the computation. Then, we add a factor of $n^3$ to the expected number of interactions taken by the Monte Carlo protocol to make it Las Vegas.

As the Monte Carlo protocol fails with probability at most $n^{-c}$ and that the expected number of interactions of the Las Vegas protocol is polynomial, the parallel expectation is still polylogarithmic.

### 4.2 The Size of the Population

The purpose of the following section is to find a way to compute the size of the population. As each agent can only contain a finite state, each agent will store one bit, and the $\log n$ first agents (according to their identifiers) will ultimately have the size written in binary when you align their bits according to their order. This way to encode an input size was also used in [9].

The protocol uses a sub-protocol that computes the median identifier of a given subset of agents. Used on the whole population, we get the first bit of the size (depending on if we have the same number of identifiers bigger and smaller to this identifier or not). We can then work on half the population. We iterate the protocol on the new half to get a new bit and a new half.

**Theorem 3.** *Finding the median identifier can be done in a polylogarithmic number of parallel interactions. The median identifier is the identifier $Med$ such that:*

$$|\{id : id \leq Med\}| - |\{id : id > Med\}| \in \{0, 1\}$$

*Proof.* We will give an idea here of the protocol.

The protocol works by dichotomy. It keeps and updates two identifiers $Min$ and $Max$ that bounds the median identifier. Here is a quick description of the steps of the protocol:

1. We initialize $Min$ and $Max$ by finding through an epidemic the smallest and the highest identifier present in the population.
2. The leader takes at random an identifier $Cand$ in $]Min, Max[$, by picking the first identifier in the interval it hears about (spreading the search of such an identifier and the reception takes two epidemics).
3. The leader performs the predicates $|x_{\leq Cand} - x_{>Cand}| = 0$, $|x_{\leq Cand} - x_{>Cand}| = 1$ and $|x_{\leq Cand} - x_{>Cand}| \geq 2$, using protocols from [4] (see Theorem 2), where $x_{\leq Cand}$ is the number of agents with an identifier smaller or equal to $Cand$ and $x_{>Cand}$ is the number of agents with an identifier higher than $Cand$.

– If the answer is $True$ for one of the two first predicates, $Cand$ is the median identifier. The algorithm is over.
– If the answer for the third predicate is $True$, we have the following inequality: $Min < Cand < Med < Max$. We replace $Min$ with $Cand$ and go back to Step 2.

– Else, we know that $Min < Med < Cand < Max$. We replace $Max$ with $Cand$ and go back to Step 2.

We proved that there is a probability greater than $\frac{1}{4}$ to divide by $\frac{4}{3}$ the number of identifiers in the interval $]Min, Max[$ after one loop of the algorithm. This permits to conclude that this algorithm will take $O(\log n)$ expected iterations.

Each iteration using $O(n \log^5 n)$ expected interactions, we get that this protocol is in $CPPL$.

The previous protocol will be used as a tool to write the size of the population on the $\log n$ first agents. It still work on any subset of agents.

**Theorem 4.** *There exists a protocol that writes in binary on the first* $\log n$ *agents the size of the population.*

*Proof.* To build this protocol, we first adapt the previous one as follows:

– The Median protocol can be used on a segment:
Instead on working on the whole population, we accept to launch it with two identifiers $A$ and $B$. We will look on the median identifiers among those who are in $[A, B]$.
– The protocol needs to check if the number of agents in the segment $[A, B]$ is even or odd. This corresponds to check whether the integer $|\{A \leq id \leq Med\}| - |\{B \geq id > Med\}|$ is equal to 0 or 1.

Each agent stores a bit $Size$ set to 0. The bits of the size are computed from the right to the left as follows:

1. Let $min$ (resp. $max$) be the smallest (resp. higher) identifier present in the population. We initialize $A$ and $B$ with, respectively, $min$ and $max$. We also initialize an identifier $C$ to $min$, it will represent the cursor pointing to which agent we write the bit of the size of the population when it is computed.
2. We compute $Med$, the median agent in $[A, B]$, and write the parity on the bit $Size$ of agent $C$.
3. We update the identifiers as follows: $B \leftarrow Med$, $C \leftarrow Next(C)$.
4. If $A \neq B$, we come back to step 2, else the computation is over.

When this protocol is over, we have

$$n = \sum_{i=0}^{\log n} 2^i Size_{Next^i(min)}.$$

where $Size_{Next^i(min)}$ is the bit $Size$ of the $(i+1)$th agent.

The Median protocol will be iterated exactly $\log n$ times. This concludes the proof.

## 5 Impossibility Results

In this section, we provide two results that motivate the idea that the population cannot take into consideration precisely the sub-words in the population (and hence, focus locally on the input). More precisely, only a polylogarithmic number of agents may know what there is exactly on their "neighbors". It is supported by the fact that only a polylogarithmic number of agents will know the identifier next of their own (Theorem 5).

The proof that Linearly Local Languages (see Definition 6) are not in $CPPL$ (Theorem 6) is based on the fact that there is a pair of consecutive identifiers such that, with high probability, the population will not be able to differentiate them, as these identifiers will not appear in a common interaction during the computation.

**Theorem 5.** *Any population protocols needs at least $\Omega(n\sqrt{n})$ expected interactions until each agent has found its Next.*

We bring now another impossibility result. We show that Community Protocols cannot link a linear number of consecutive identifiers in $CPPL$. To prove this, we introduce a new class of languages:

**Definition 6.** *Let $u = u_1 \dots u_N$ a word of size $N$ and $i < N$. We call $\sigma_i(u)$ the word $u$ where the $i$th letter is permuted with the next one. More formally, we have:*

$$\sigma_i(u) = u_1 \dots u_{i-1} u_{i+1} u_i u_{i+2} \dots u_N$$

*We say that a language $L$ is* Linearly Local *if there exists some $\alpha \in ]0, 1]$ such that, for any $n$, there exists some $u \in L$ and some $I \subset \mathbb{N}$ such that:*

*$u = u_1 \dots u_N$ with $N \geq n$, $\exists I \subset [1, N-1]$, $|I| \geq \alpha N$ and for all $i \in I$, $\sigma_i(u) \notin L$.*

These languages are said linearly local as, for any size of input, we can find words that have a linear number of local regions where a small permutation of letter leads to a word not in the language.

**Theorem 6.** *There is no linearly local language in $CPPL$.*

To prove this result, the idea is to prove that for any protocol, and for any $n$, there exists some $u$ in the language of length at least $n$ and $i \in I$ such that there is a high enough probability that the protocol acts the same way on the inputs $u$ and $\sigma_i(u)$.

Let $\alpha \leq 1$, and let $(I_n)_{n \in \mathbb{N}}$ be a sequence such that, for any $n \in \mathbb{N}$, we have $I_n \subset [1, n]$ and $|I_n| \geq \alpha n$.

We work on pairs $(Id_i, Id_{i+1})_{i \in I_n}$. We want to prove that, for any $n$, there is some $i \in I_n$ such as, with high probability, the identifiers $Id_i$ and $Id_{i+1}$ never appeared in the same interaction after any $n$-polylog number of interactions. In the proof, $Id_i$ meets $Id_{i+1}$ means both identifiers appear in the slots of two interacting agents when the interaction occurs.

To prove that, we first introduce the 3 following lemmas. Only the last one will be proved.

**Lemma 1.** *Let $f$ a $n$-polylog function and let $\alpha > 0$.*
*To each identifier id, we define the set $E_{id}$ and value $M_{id}$ as*

- $E_{id} = \{$*Agents having had id in one of its register after* $f(n)$ *steps*$\}$
- $M_{id} = |E_{id}|$.

*There exists some polylogarithmic function $g$ such that, for $n$ large enough, after $f(n)$ steps:*

$$\mathbb{E}(|\{id : M_{id} \leq g(n)\}|) \geq \left(1 - \frac{\alpha}{2}\right) n$$

With this first result, we deduce that at most a small fraction of the pairs $(Id_i, Id_{i+1})$ could have met after $n$-polylog number of steps. This means that $Id_i$ and $Id_{i+1}$ never appeared in the slots of two agents that interacted together, when they interacted.

**Lemma 2.** *Let $f$ be a $n$-polylog function. For $n$ big enough, after $f(n)$ steps:*

$$\mathbb{E}(|\{i \in I_n : Id_i \text{ and } Id_{i+1} \text{were in a same interaction}\}|) \leq \frac{3}{4}\alpha n$$

**Lemma 3.** *Let $f$ be a $n$-polylog function. For any $n$ large enough, there exists $i \in I_n$ such as:*

$$\Pr(Id_i \text{ met } Id_{i+1}) \leq \frac{3}{4}$$

*Proof.* Let suppose that for any $i$, $\Pr(Id_i \text{ met } Id_{i+1}) \geq \frac{3}{4}$.
That implies, $\mathbb{E}(N) = \sum_{i \in I_n} \Pr(Id_i \text{ met } Id_{i+1}) \geq \frac{3}{4}\alpha n$.
This is a direct contradiction of the previous lemma.

To prove our theorem, we need to prove the following proposition:

**Proposition 4.** *For any protocol, for any $n$-polylog function $f$, for any input of size $n$ large enough, there exists some $i \in I_n$ such that the probability that the identifiers $Id_i$ and $Id_{i+1}$ never appeared on a same interaction after $f(n)$ steps is greater than $\frac{1}{4}$.*

This proposition is a direct corollary of previous lemma. With this proposition, the proof of Theorem 6 can be done as follows:

*Proof.* Let $L$ be a linearly local language with parameter $\alpha > 0$. Let $P$ be a protocol computing $L$ in less than $n \log^m n$ expected interactions for some $m \in \mathbb{N}$. Let choose $n$ large enough to have the property of Proposition 4 with $f(n) = 9n \log^m n$. Let $u$ be a word of size $N \geq n$ such that the corresponding $I$ has a size greater than $\alpha N$.
We have, from Markov's Inequality that:
$\Pr(\text{number of steps to compute } u \leq 9N \log^m N) \geq \frac{8}{9}$.
It implies that at least $\frac{8}{9}$ of the sequences of configurations of length $9N \log^m N$ provides the right output.

By applying the previous proposition, we obtain the existence of some $i \in I$ such that the probability that the identifiers $Id_i$ and $Id_{i+1}$ never appeared on a same interaction after $9N \log^m N$ steps is greater than $\frac{1}{4}$.

This implies that in at least $\frac{1}{4}$ of the sequences of configurations of length $9N \log^m N$, $Id_i$ and $Id_{i+1}$ were never in a common interaction.

Hence, if $Id_i$ and $Id_{i+1}$ never appear on a same interaction, then $P$ will not see any difference between the two inputs $u$ and $\sigma_i(u)$.

Between the $\frac{8}{9}$ of the sequences that provides the right output on these two inputs, at least $\frac{7}{9}$ are common (two sequences are here said to be common if the sequence of the interacting identifiers are equals).

As $\frac{7}{9} \geq 1 - \frac{1}{4}$, amongst those common sequences, some of them does not involve $Id_i$ and $Id_{i+1}$ in a same interaction. As the protocol cannot differentiate those two inputs during those sequences, it cannot bring the right output.

This provides a contradiction. There is no protocol in $CPPL$ that computes $L$.

**Corollary 2.** *The rational language $(ab)^*$, the rational language of words not containing the subword $(ab)$, the well-formed parenthesis language and the palindrome language are not in $CPPL$.*

*Proof.* For the first language, to each $n$ we can associate $u = (ab)^n$, with $\alpha = 1/2$. Same thing with the third one, replacing $a$ with the opening parenthesis and $b$ with the closing one. For the fourth, $(ab)^n a$ works the same way. Finally, for the second one, $(bac)^n$ and $\alpha = 1/3$ works.

## 6 Set Considerations

We provide finally, in this section, set comparisons with $CPPL$. We first give a large upper bound:

**Theorem 7.**

$$CPPL \subset NSPACE(n \log n) \cap \bigcup_{k \in \mathbb{N}} SPACE(n \log^k n)$$

We now provide a class of Turing Machines that computes everything we found to be computable yet. This class of machines is capable of computing global properties, through the ability to work on subsets of agents. It is capable to compute the size of sets of agents. It can perform any polylogarithmic number of steps of a regular Turing Machine.

This machines are capable of focusing only on a polylogarithmic regions of agents. It motivates the belief that Community Protocols are not capable of local knowledge on too much places.

**Theorem 8.** *Let $M_T$ a Turing Machine on alphabet $\Gamma$ recognizing the language $L$ having the following restrictions. There exists some $k \in \mathbb{N}$ such that*

- *$M_T$ has 4 tapes. The first one is for the input $x$.*
- *The space of work is restricted as follows:*
  - *The first tape uses only the input space of $|x|$ cells.*
  - *The 2nd and the 3rd use at most a space of $\log |x|$ cells.*
  - *The 4th uses at most a space of $\log^k |x|$ cells.*
- *$M_T$ can only do at most $\log^k |x|$ unitary operations among the following ones:*
  1. *A regular Turing Machine step.*
  2. *Mark/Unmark the cells that have the symbol $\gamma \in \Gamma$.*
  3. *Write in binary on the 2nd tape the number of marked cells.*
  4. *Go to the cell of the number written on the 3rd tape if this number is smaller than $|x|$.*
  5. *Mark/Unmark all the cells left to the pointing head on the first tape.*
  6. *Turn into state $\gamma'$ all the marked cells in state $\gamma \in \Gamma$.*
  7. *Select homogeneously a random number between 1 and the number written on the 3rd tape if this number is smaller than $|x|$.*

*Then we have $L \in CPPL$.*

## References


1. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. DISC **20**, 279–304 (2007)
2. Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 63–74. Springer, Heidelberg (2005). https://doi.org/10.1007/11502593_8
3. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Principles of Distributed Computing, PODC, July 2004
4. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distrib. Comput. DISC **21**, 183–199 (2008)
5. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing population protocols. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 103–117. Springer, Heidelberg (2006). https://doi.org/10.1007/11795490_10
6. Beauquier, J., Blanchard, P., Burman, J., Delaët, S.: Computing time complexity of population protocols with cover times - the zebranet example. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 47–61. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24550-3_6
7. Beauquier, J., Burman, J., Kutten, S.: Making population protocols self-stabilizing. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 90–104. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05118-0_7
8. Beauquier, J., Burman, J., Rosaz, L., Rozoy, B.: Non-deterministic population protocols. In: Baldoni, R., Flocchini, P., Binoy, R. (eds.) OPODIS 2012. LNCS, vol. 7702, pp. 61–75. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35476-2_5
9. Bournez, O., Cohen, J., Rabie, M.: Homonym population protocols. In: Bouajjani, A., Fauconnier, H. (eds.) NETYS 2015. LNCS, vol. 9466, pp. 125–139. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26850-7_9

10. Chatzigiannakis, I., Michail, O., Nikolaou, S., Pavlogiannis, A., Spirakis, P.G.: Passively mobile communicating machines that use restricted space. In: International Workshop on Foundations of Mobile Computing, FOMC 2011 (2011)
11. Daley, D.J., Kendall, D.G.: Stochastic rumours. IMA J. Appl. Math. **1**, 42–55 (1965)
12. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: When birds die: making population protocols fault-tolerant. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 51–66. Springer, Heidelberg (2006). https://doi.org/10.1007/11776178_4
13. Doty, D., Soloveichik, D.: Stable leader election in population protocols requires linear time. In: Moses, Y. (ed.) DISC 2015. LNCS, vol. 9363, pp. 602–616. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48653-5_40
14. Fraigniaud, P., Korman, A., Lebhar, E.: Local MST computation with short advice. In: Symposium on Parallelism in Algorithms and Architectures, SPAA (2007)
15. Gillespie, D.T.: A rigorous derivation of the chemical master equation. Phys. A **188**, 404–425 (1992)
16. Guerraoui, R., Ruppert, E.: Names trump malice: tiny mobile agents can tolerate byzantine failures. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 484–495. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02930-1_40
17. Hethcote, H.W.: The mathematics of infectious diseases. SIAM Rev. **42**, 599–653 (2000)
18. Mertzios, G.B., Nikoletseas, O.E., Raptopoulos, C.L., Spirakis, P.G.: Stably computing order statistics with arithmetic population protocols. In: Mathematical Foundations of Computer Science, MFCS (2016)
19. Michail, O., Chatzigiannakis, I., Spirakis, P.G.: Mediated population protocols. Theor. Comput. Sci. **412**, 2434–2450 (2011)
20. Murray, J.D.: Mathematical Biology. I: An Introduction, 3rd edn. Springer, Heidelberg (2002). https://doi.org/10.1007/b98868
21. Schönhage, A.: Storage modification machines. SIAM J. Comput. **9**, 490–508 (1980)

# On the Smallest Grain of Salt to Get a Unique Identity

Peva Blanchard(✉) and Rachid Guerraoui

EPFL, Lausanne, Switzerland
{peva.blanchard,rachid.guerraoui}@epfl.ch

**Abstract.** We study the fundamental *enumeration* problem in asynchronous message-passing networks. Anonymous processes have to eventually decide on pairwise distinct identifiers, despite all starting in the same initial state. It is known since Angluin's seminal result [2] that some *grain of salt* is required for distributed algorithms to solve the problem, e.g., the system needs to have a non-symmetrical topology or unbiased independent random bits.

The starting point of this paper is the observation that these approaches demand too strong assumptions. In short, by adding *time* to the picture, we show that the enumeration problem can be solved with far less. The idea is to consider a schedule of events in a distributed system as a *space-time structure* that is *gradually learnt* by the processes. We introduce the notion of *divergence time* which essentially measures the time by which the causal order induced by the system schedule has differentiated all the processes.

We prove lower bounds on the running time of any algorithm solving enumeration in terms of divergence time. In particular, we show that any adversary scheduler against which the enumeration problem can be solved *necessarily* selects schedules with *finite divergence time*.

We prove that this last condition is *sufficient*: we present the Torche algorithm which solves enumeration for all schedules with finite divergence time. In this sense, having finite divergence time is the smallest grain of salt required to solve the enumeration problem.

## 1 Introduction

Process identifiers are crucial in distributed systems, and are implicitly assumed in most distributed algorithms. The problem of assigning distinct identifiers to processes, however, is not trivial. This problem, called *enumeration*, has received a lot of attention in the past decades [1,2,6,9,11,16,21–23].

The problem consists for a set of $n$ processes, starting identically (in the same initial state), to each decide a value (an integer in the range $\{1,\ldots,n\}$) that is different from all the values decided by the other processes. The core difficulty has been pinpointed in Angluin's seminal paper [2]. In short, the problem is impossible to solve deterministically in any network with *spatial symmetry*. Roughly speaking, two processes in the network are said to be related by a spatial symmetry if they have the exact same view of their surrounding. To get


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 106–121, 2017.
https://doi.org/10.1007/978-3-319-72050-0_7

an intuition of Angluin's argument, consider a deterministic algorithm running on an even-sized oriented network of processes. Assume, for the sake of simplicity, that when a process is activated, the process atomically reads the states of all its neighbours in the network, and updates deterministically its own state accordingly. Initially, all the processes have the same state. The neighbourhood of any process $p$ is thus similar to the neighborhood of the diametrically opposite process $q$. If $p$ and $q$ are concurrently activated, then their states are updated to the same value. By repeating this kind of activation, one can design schedules of events during which no process is distinguishable from the opposite process, thus preventing enumeration.

The only way to circumvent Angluin's argument is to assume that the distributed system contains some *grain of salt*, i.e., that there is some breaking of symmetry somewhere which could be exploited by distributed algorithms to enumerate. So far, two sorts of grain of salts have been considered in the literature. One approach *(I)*, adopted in [6,9,16], assumes that the *topology of the network has no non-trivial spatial symmetries*. Another approach *(II)* is the use of randomization [1,11], with processes having local access to *unbiased independent* random bits.

The starting point of this paper is the observation that these approaches are not necessary. The idea is to consider a schedule of events in a distributed system as a *space-time structure* that is *gradually learnt* by the processes.

Instead of considering that only the spatial part (i.e. the network topology) has no non-trivial symmetries, we consider the much weaker assumption that the space-time structure, taken as a whole, has no non-trivial symmetries. Our approach encompasses naturally the two previous approaches. First, if the spatial part has no non-trivial symmetries, then the space-time structure necessarily lacks non-trivial symmetries; second, the use of independent random bits ensures, with probability one, that the space-time structure lacks non-trivial symmetries.

We consider a general asynchronous distributed system model where processes communicate with their neighbours by sending and/or receiving messages through communication channels. Moreover, the processes may have access to unreliable[1] sources of randomness. The different possibilities of scheduling events are chosen by some external entity, called the *adversary scheduler*.

Our main conceptual contributions are twofold. First, we introduce the notion of *divergence time* of a schedule $S$ of events, which, roughly speaking, measures the time by which all the processes have differentiated[2]. We then exhibit lower bounds on the running time of any algorithm solving the enumeration problem, in terms of divergence time. In particular, we show that any adversary scheduler against which it is possible to solve the enumeration problem can only select schedules that have *finite divergence time*. In other words, the finite divergence time condition is a *necessary* condition for solving the enumeration problem.

[1] The bits used by the processes may, to some extent, be correlated, across the network and through time.

[2] More precisely, they have pairwise non-isomorphic causal pasts. See details below.

Second, we prove that the finite divergence time condition is actually *sufficient*. We present an algorithm, we call Torche, that solves the enumeration problem over *all* the schedules with finite divergence time, assuming only the knowledge of the network size[3].

The main consequence of the existence of this algorithm is that the finite divergence time condition is indeed the *smallest grain of salt* required to solve the enumeration problem. By "smallest grain of salt", we mean that any other grain of salt, i.e., any other property of the schedules which allows to break symmetry and solve enumeration (e.g., topology without symmetries, or use of randomness) necessarily implies that the divergence time is finite.

Two new techniques are involved in the design of Torche: *folded causal past reconstruction* and *phylogenetic tree extraction*.

- First, the *folded causal past reconstruction* technique consists for each process $p$ to maintain a compressed estimate of its causal past, by gluing together the estimates of its neighbours, and folding together the events that have isomorphic causal pasts. This technique may be seen as the spatio-temporal generalization of the *compressed view* technique from [20].
- Second, Each process $p$ can extract from the folded estimate of its causal past a *phylogenetic tree* which, roughly speaking, represents the various differentiations that have occurred, as far as process $p$ knows. The number of vertices that lies in the same level in this tree somehow gives the *effective* number of distinct processes, i.e., the number of distinguishible (groups of) processes. Assuming that the schedule has finite divergence time, this tree eventually has $n$ branches, where $n$ is the network size. Process $p$ can then detect the end of this divergence period, and, since $p$ knows on which branch of the tree it lies, process $p$ can also decide on a unique identifier.

Our Torche algorithm has several interesting properties. The running time is tight in the sense that processes decide right after the divergence time plus the time for the information at any process to reach the whole network (cover time). The space required for storing the process states and messages is polynomial in the network size, the divergence and cover time.

**(Paper organization).** We present our computational model, as well as the notions of divergence time in Sect. 2. We present our lower bounds on the running time of any algorithm solving the enumeration problem in Sect. 3. In Sect. 4, we present the Torche algorithm, and prove its main properties. In Sect. 5, we show how our notion of divergence time encompasses other notions used to circumvent Angluin's argument in previous work, namely *fiber-minimal* networks, and the *use of random bits*. Finally, we discuss the related work in Sect. 6.

## 2 Model and Definitions

We consider a general asynchronous model of computation where anonymous processes communicate by message passing.

[3] The knowledge of the network size, or any similar property, is a common assumption in the literature [21,22].

### 2.1 Algorithms

**(Graphs).** We consider directed graphs with (possibly) multiple arrows between two vertices, and (possibly) self-loops. Formally, a *graph* $G$ is given by a set $\mathcal{V}G$ of vertices, a set $\mathcal{A}G$ of arrows, and maps $s, t : \mathcal{A}G \to \mathcal{V}G$ specifying the source and target vertices of each arrow. A *vertex-labeling* (resp. *arrow-labeling*) is a map $\mathcal{V}G \to X$ (resp. $\mathcal{A}G \to \Lambda$). A *path* is a sequence $a_1 \dots a_l$ of arrows such that $t(a_i) = s(a_{i+1})$ for $1 \leq i < l$. This path is a *cycle* if moreover $t(a_l) = s(a_1)$. The graph $G$ is *acyclic* if it does not contain any cycle.

A *morphism* $\phi : G \to H$ is given by a vertex function $\phi_V : \mathcal{V}G \to \mathcal{V}H$, and an arrow function $\phi_A : \mathcal{A}G \to \mathcal{A}H$ such that $s(\phi_A(a)) = \phi_V(s(a))$ and $t(\phi_A(a)) = \phi_V(t(a))$ for every arrow $a \in \mathcal{A}G$. If moreover the graphs $G$ and $H$ are equipped with a vertex-labeling and an arrow-labeling, it is also required that $\phi_V$ and $\phi_A$ preserve the labels. The morphism $\phi$ is an *isomorphism* if both $\phi_V$ and $\phi_A$ are bijective. We denote by $G \simeq H$ the statement that $G$ and $H$ are isomorphic.

**(Networks).** A *network* is $\mathcal{N}$ simply modeled as a graph with at most one arrow from one vertex to another, and without self-loops. The vertices represent the processes. An arrow with source $p$ and target $q$ represents a communication channel transporting messages from $p$ to $q$. For each process $p$, we denote by $\mathcal{N}_p$ the set of neighbour processes with arrow towards process $p$.

**Algorithm 1.** Normal Form - process $p$

```
1 variables:
2     state s_p of p initially set to some common value
3     set t_p of triples (ω, z) where ω ∈ Λ_p and z is a state value
4     variable b_p for random bits
5 for round r = 0, 1, ...
6     t_p ← scan()                                  /* scan neighbours */
7     b_p ← rand()                                  /* ''random'' bits */
8     s_p ← new-state(s_p, t_p, b_p)                   /* update state */
```

**(Processes).** The processes are anonymous (no identifiers), they start in the same initial state, and all execute the same algorithm. We also assume that each process can read some bits from some local source of information. These bits can be thought as "random" bits, but we insist on the fact that, in the most general case, these bits may be correlated (both in time and across the network) in any possible way.

Each process $p$ performs an infinite series of *asynchronous* rounds. Each round comprises two phases: a *scan* phase, and an *update* phase. The scan phase collects the (possibly not up to date) states of its neighbours and returns the multiset $t_p$ of couples $(\omega, z)$ where $z$ is the state of a neighbour $q$ received along the incoming arrow $a$ with label $\omega$ and source $q$. Then, during the update

phase, process $p$ reads some bit-string $b_p$ from its local source of information, and updates its state by applying a deterministic transition function `new-state` to the tuple formed by its current state, the multiset $t_p$ collected during the scan phase, and the bit-string $b_p$. Algorithm 1 sums up the normal form of an algorithm.

### 2.2 Schedules

We model a (finite or infinite) schedule $S$ on a network $\mathcal{N}$ as a (finite or infinite) acyclic graph equipped with a labeling of the vertices with bit-strings, and a labeling of the arrows with the same set of arrow labels as the network $\mathcal{N}$. The vertices of the schedule are couples $(p, r)$ where $p$ is a process, $r \geq 0$ is an integer. We refer to $(p, r)$ as an *event* at $p$ in $S$. Each $(p, r)$ with $r \geq 1$ is labeled with a bit-string $b$ representing the "random" bit-string read by $p$ during round $r - 1$. Intuitively, each event $(p, r)$ represents the state of $p$ at the beginning of round $r$ at $p$. This state is the result of the previous update phase at round $r - 1$, and depends on the previous state of $p$, as well as the state values collected during the previous scan operation. These dependencies are modeled by arrows between events.

More precisely, if during round $r$, process $p$ received the state associated with the round $s$ of the neighbour $q$, then there is an arrow $(q, s) \rightarrow (p, r)$ labeled with the same label as the arrow from $q$ to $p$ in the network. For every $r \geq 0$, there is also an arrow $(p, r) \rightarrow (p, r + 1)$ labeled with the distinguished symbol $\epsilon$. Figure 1 gives an example of a schedule.

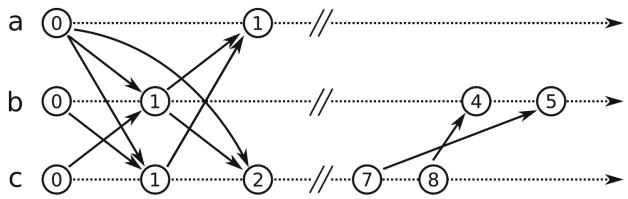



**Fig. 1.** Schedule - process $c$ in round 2 reads the initial state of process $a$, and the state at the beginning of round 1 of process $b$. Note (on the right) that a scan may return values older than the values returned by the previous scan. The labels of the arrows are omitted.

**(Causality).** For any two events $e, e'$ in $S$, $e$ *causally precedes* $e'$ in $S$ if there exists a path from $e$ to $e'$ in $S$. The *causal past* of an event $e$ in $S$ is the schedule which is the sub-graph of $S$ spanned by $e$ and all the events causally preceding $e$ in $S$. Given a finite schedule $S$, for any process $q$ participating in $S$, we refer to the causal past of the latest event at $q$ in $S$ as the *causal past of $q$ in $S$*. A *past cone atp* is a finite schedule $P$ which is the causal past of some event $e = (p, r) \in \mathcal{V}P$. The event $e$ is necessarily unique, and is referred to as the *apex*

of $P$. The *causal height*, or simply *height*, of an event $e$ in $S$ is the length of the longest directed path in $S$ reaching $e$. The height of a finite schedule $S$ is the maximum height of its events, i.e., the length of the longest directed path contained in $S$. Note that the height of a past cone is the height of its apex.

**(Cut).** A *cut* $C$ of a schedule $S$ is a set of events in $S$ such that each process of the network participates exactly once in $C$. For each process $p$, we denote by $C[p]$ the height of the event $(p, r)$ in $C$. The *initial cut* is defined by $\{(p, 0) \mid p \in \mathcal{N}_V\}$. Cuts are partially ordered: $C \preccurlyeq C'$ if, for all $p$, $C[p] \leq C'[p]$. Equipped with this partial order, the set of cuts of a schedule form a lattice.

**(Segment).** Given two cuts $C \preccurlyeq C'$ in a schedule $S$, the *segment* $K = [C, C']$ is the sub-schedule of $S$ comprising all the events $(p, r)$ with height $C[p] \leq h \leq C'[p]$, where $p$ runs over all the processes. The height of the segment $K$ is the length of the longest causal chain in $K$. The *prefix* $[0, C]$ is the sub-schedule of $S$ comprising all the events $(p, r)$ with height at most $C[p]$. The *suffix* $[C, \infty)$ is the sub-schedule of $S$ comprising all the events $(p, r)$ with height at least $C[p]$.

**(Fairness).** The schedule $S$ is *fair* if, for any two processes $p$ and $q$, for all $t \geq 1$, there is a cut $C$ such that $\forall p, C[p] \geq t$ and $[C, \infty)$ contains a path from some event at $p$ to some event at $q$. Unless stated otherwise, all infinite schedules are assumed to be fair.

**(Adversary scheduler).** An *adversary scheduler* is modeled as a set of (fair)infinite schedules. Adversary $\mathcal{A}$ is *stronger* than adversary $\mathcal{B}$ if $\mathcal{B} \subseteq \mathcal{A}$.

**(Divergence cut).** The *divergence cut* of a schedule $S$ is the minimum cut $C_{dv}(S)$ such that, for all cuts $C \succcurlyeq C_{dv}(S)$ in $S$, the causal pasts of any two distinct events in $C$ are not isomorphic. The *divergence time* of a schedule $S$ is defined as $\tau_{dv}(S) = \max_p C_{dv}[p]$. If the divergence cut is undefined, we write $C_{dv}(S) = \infty$ and $\tau_{dv}(S) = \infty$. We say that the schedule $S$ has finite divergence time if the divergence cut exists. When it is clear from the context, we simply write $C_{dv}$ and $\tau_{dv}$, omitting the reference to the schedule $S$.

**(Cover cut function).** The *cover cut function* of a schedule $S$ is defined as follows. For any cut $C$ in $S$, $C_{cv}(C, S)$ is the minimum cut $C'$ such that, for any two processes $p, q$, there is a path in $[C, C']$ from some event at $p$ to some event at $q$. When it is clear from the context, we simply write $C_{cv}(C)$, omitting the reference to the schedule $S$.

**(Decision event).** Any process $p$ is assumed to be able to trigger a *decide* action. Afterwards, its state remains unchanged. It is assumed that each process can trigger this action at most once during the execution. The event at $p$ at which this action is performed is the *decision event at* $p$.

**(Decision cut).** The *decision cut* is the cut formed by the decision events of all the processes. If some process never decides, the decision cut is undefined.

## 3 Lower Bounds on the Running Time

In this section, we present lower bounds on the (causal) height of the decision events of the processes, in terms of divergence and cover cuts. The following proposition states that any adversary scheduler against which enumeration is solvable *necessarily* provides schedules with finite divergence time. More precisely, it is impossible for all processes to decide strictly before the divergence cut.

**Proposition 1.** *Let $\mathcal{A}$ be a set of schedules over the network $\mathcal{N}$. Consider an algorithm solving enumeration over all the schedules in $\mathcal{A}$. Then all schedules in $\mathcal{A}$ have finite divergence time. And, more precisely, for any schedule in $\mathcal{A}$, the cut $C$ defined by the decision events satisfies*

$$\exists p \in \mathcal{N},\ C[p] \geq C_{dv}[p]$$

*Proof.* Consider a schedule $S$ in $\mathcal{A}$. Let $C$ denote the decision cut. Assume first that the divergence cut of $S$ is undefined. By definition, this implies that there exists a cut $D \succcurlyeq C$, and two distinct events in $D$, at two distinct processes $p, q$, with isomorphic causal pasts. In particular, $p$ and $q$ must have decided on the same value; whence a contradiction. Thus, $S$ has finite divergence time.

Assume now that $C \prec C_{dv}$, i.e., for all processes $p$, $C[p] < C_{dv}[p]$. By definition of the divergence cut, this means that there exist two distinct processes $p, q$ such that the causal pasts of their decision events are isomorphic. This implies that $p$ and $q$ decide on the same value; whence a contradiction. □

The following states that, in general, at least some process has to wait until it notices that all processes have differentiated.

**Proposition 2.** *Consider an algorithm solving enumeration. Then there exists a network $\mathcal{N}$ and a schedule on $\mathcal{N}$ such that the cut $C$ defined by the decision events of the processes satisfies*

$$\exists p \in \mathcal{N},\ C[p] \geq C_{cv}(C_{dv})[p]$$

*Proof.* We consider a linear network $\mathcal{N}$ of 3 vertices, and the schedules $S_1$ and $S_2$ in Fig. 2. The figure also depicts the divergence and cover cuts for schedule $S_1$. We claim that, in $S_1$, process $c$ cannot decide before its second event. Indeed, assume that process $c$ decides at its first event. However, at this point, $S_1$ and $S_2$ are indistinguishable for process $c$. In $S_2$, the first events of $a$ and $c$ have isomorphic causal pasts. Thus, in $S_2$, process $a$ decides on the same value as $c$; whence a contradiction. In particular, the decision cut $C(S_1)$ in $S_1$ satisfisfies $C(S_1)[c] \geq C_{cv}(C_{dv}, S_1)[c]$. □

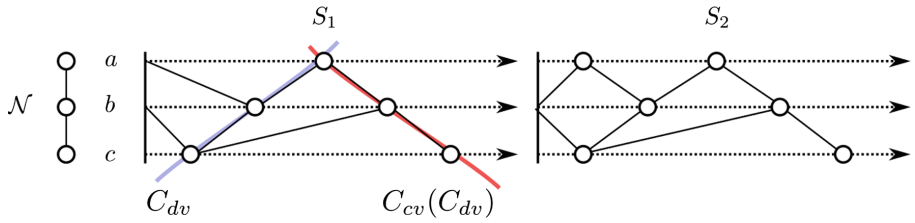



**Fig. 2.** Proof of Proposition 2 - process $c$ cannot decide before its second event.

## 4 The Torche Algorithm

Proposition 1 states that, in order to solve the enumeration problem, it is necessary that the schedules chosen by the adversary scheduler have a finite divergence time. In this section, we prove that having a finite divergence time is a sufficient condition for solving the enumeration problem. In this sense, having a finite divergence time is indeed the smallest grain of salt required to solve the enumeration problem.

The main idea underlying our approach consists in having processes trying to get an estimate, as accurate as possible, of their causal pasts. For the sake of simplicity, let us consider how a naive version of a full information algorithm would work. Process $p$ starts in its initial state, represented as a single vertex. In the next round, process $p$ collects the states of (some of) its neighbours, draws a random value $b$ (if any), and updates its own state. In a naive full-information algorithm, this new state is encoded as a tree. The root corresponds to the new round that $p$ has just performed, and is labeled with the value $b$ that has been drawn. The root's children are the trees received from the neighbours, as well as the tree corresponding to the previous state of $p$. If $p$ has received the (tree) state of $q$ along the incoming arrow $q \xrightarrow{\omega} p$, then the arrow connecting the tree of $q$ to the root is labeled with $\omega$ as well. The arrow connecting the previous tree of $p$ to the root is labeled with the distinguished symbol $\epsilon$. The first row in Fig. 4 illustrate how the full-information trees evolve along the schedule of Fig. 3.

An *$\epsilon$-path* is a path whose edges are all labeled with $\epsilon$. A *maximal$\epsilon$-path* is an $\epsilon$-path which cannot be extended to a longer $\epsilon$-path. A first observation is that, from the designer's point of view, the maximal $\epsilon$-paths can be mapped to process identities. However, the tree structure has a lot of redundancy. For example, if $p$ sees $q'$ which has seen $p$ before, then there are two maximal $\epsilon$-paths that can be associated with $p$. For instance, in Fig. 4, the tree $T_p^3$ contains two copies of $T_p^1$.

This example leads to a second observation: the tree of $p$ that was observed by $q$ is isomorphic to a sub-tree of the latest tree of $p$. A third observation is that there is one-to-one correspondence between such trees and the causal pasts that led to them. In particular, two processes have isomorphic causal pasts if and only if the corresponding trees are isomorphic.

From these observations, we adopt the following approach. We *fold* the tree by identifying vertices whose sub-trees are isomorphic. This amounts to identifying

events in the schedule that have isomorphic causal pasts. Let $W$ be the graph obtained this way. Since it is possible that two distinct processes have lived isomorphic causal pasts, this folding operation may identify them. See the second row in Fig. 4. Moreover, the $\epsilon$-paths in $W$ form a tree we call the *phylogenetic tree* associated with $W$. This phylogenetic tree is interpreted as follows. Initially, all the processes are in the same initial state, so the *effective number* of distinct processes is 1, and is encoded by the fact that the $\epsilon$-tree has a single root. As time flows, processes undergo specific events that differentiate them, and the tree branches. The number of leaves of this tree gives the effective number of distinct processes at the end of the schedule, or, mathematically speaking, the number of isomorphism classes of maximal causal pasts in the schedule. In particular, the number of leaves is at most $n$. By the assumption that the divergence time is finite, any two processes eventually have non isomorphic causal pasts, and thus, the number of leaves eventually reaches $n$.

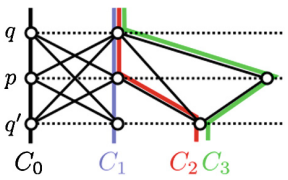



**Fig. 3.** Schedule with processes $p, q$ and $q'$. Four cuts are represented. See Fig. 4 for the states of the processes at these cuts.

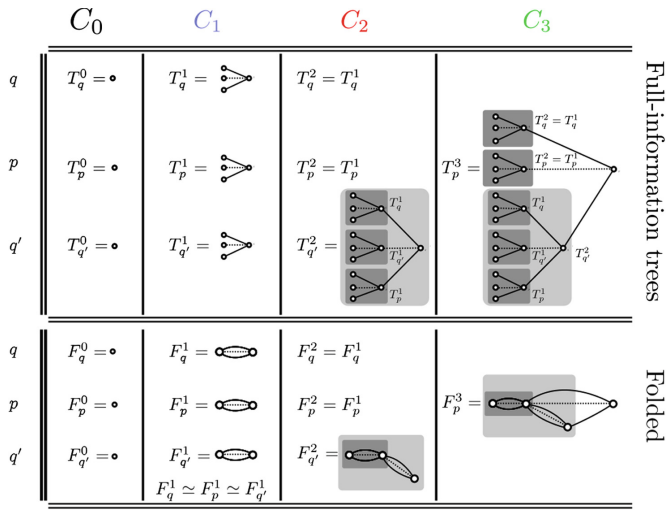



**Fig. 4.** Evolution of the full-information trees (first row) and their respective folding (second row) at the cuts from Fig. 3. Due to space limitations, at cut $C_3$, we only represent the tree and folded tree of process $p$.

As we assume that the processes know the size $n$ of the network, process $p$ can detect that all processes have differentiated (i.e., have pairwise non isomorphic causal pasts). Moreover, $p$ is aware of the leaf representing itself: it is the leaf with the highest height. Roughly speaking, process $p$ finally sorts the leaves of $W$ and decides on the rank of its leaf.

### 4.1 Fold Operation

Let $P$ be a schedule. Recall that some arrows may be labeled with the distinguished symbol $\epsilon$. There is a partial order on the vertices: $u \preccurlyeq v$ if there is a path from $u$ to $v$. We define $\downarrow v$ as the subgraph of $P$ induced by the vertices $u \preccurlyeq v$. We define an equivalence relation on the vertices: $u \sim v$ if the the graphs $\downarrow u$ and $\downarrow v$ are isomorphic. We denote by $[u]$ the equivalence class of $u$. We also define an equivalence relation on the arrows: $u \xrightarrow{\omega} v \sim u' \xrightarrow{\omega'} v'$ if there is some isomorphism between $\downarrow v$ and $\downarrow v'$ mapping the arrow $u \xrightarrow{\omega} v$ to $u' \xrightarrow{\omega'} v'$. This implies $v \sim v'$, $u \sim u'$ and $\omega = \omega'$. We denote by $[u \xrightarrow{\omega} v]$ the equivalence class of $u \xrightarrow{\omega} v$.

The graph $W = \texttt{fold}(P)$ is defined as follows. The vertices of $W$ are the equivalence classes $[u]$ with $u \in \mathcal{V}P$. The arrows of $W$ are the equivalence classes $[u \xrightarrow{\omega} v]$ with $u \xrightarrow{\omega} v \in \mathcal{A}P$. The source (resp. target) of $[u \xrightarrow{\omega} v]$ is $[u]$ (resp. $[v]$). The complexity of the fold operation is related to the complexity of graph isomorphism. Isomorphism of trees can be tested in linear time. However, the same problem for directed acyclic graphs is equivalent to the classic (undirected) graph isomorphism problem. By the recent work of [4], $W$ can be computed from $P$ in quasi-polynomial time.

**Lemma 1.** *The map $\Phi : P \to \texttt{fold}(P)$ defined on the vertices by $u \mapsto [u]$ and on the arrows by $u \xrightarrow{\omega} v \mapsto [u \xrightarrow{\omega} v]$ is a graph morphism preserving the labels of the vertices and arrows. Moreover, $\Phi$ also preserves the height of the vertices.*

*Proof.* The claim follows from the definition of $\texttt{fold}(P)$. □

**Lemma 2.** *Let $P$ be a schedule. Let $W = \texttt{fold}(P)$. The number of maximal $\epsilon$-paths in $W$ equals the number of isomorphism classes of the maximal causal pasts of the processes in $P$. In particular, this number is at most the number $n$ of processes in the network.*

*Proof.* Let $E$ denote the set of maximal events in $P$, i.e., the events $e$ which do not causally precede any other event. The set $E$ can be partitioned into equivalence classes. Each maximal $\epsilon$-path $\pi$ in $W$ can be identified with the last vertex $l$ along this path (equivalently, the corresponding leaf in the tree formed by the maximal $\epsilon$-paths). By definition of $\texttt{fold}$, the vertex $l$ denotes an equivalence class of events in $P$. An event $e$ in the class $l$ is maximal, as otherwise (since the map $\Phi$ preserves the height) $l$ would not be the last vertex along $\pi$. Therefore, there is one-to-one correspondence between the equivalence classes partitioning $E$ and the final vertices of maximal $\epsilon$-paths in $W$. □

### 4.2 Algorithm Details

Each process $p$ maintains an estimate $W_p$ of its folded causal past. At the beginning of a round, process $p$ scans the set $\mathcal{N}_p$ of its neighbours, and pulls their estimates $(W_q)_{q\in\mathcal{N}_p}$. Process $p$ then forms the disjoint union of these graphs, including its previous estimate, adds a new vertex $(r, b)$ labeled with the current round number $r$ and a random value $b$ (if any), and connects the apex of each estimate $W_q$ with the new vertex $(r, b)$; this arrow being labeled with the same label $\omega$ as the one connecting $q$ to $p$. We denote this whole operation by $\left(\bigsqcup_{q\in\mathcal{N}_p\cup\{p\}} W_q\right) \oplus (r, b)$. The variable $W_p$ is updated by folding the aforementioned graph.

As will be shown below, $W_p$ is isomorphic to the folding of the causal past of process $p$. Process $p$ derives the phylogenetic tree associated with $W_p$ and counts the number $c$ of maximal $\epsilon$-paths. If process $p$ notices that $c = n$, i.e., that all processes have differentiated, then process $p$ computes the shortest prefix $K_p$ of $W_p$ whose phylogenetic tree still has $n$ leaves (maximal $\epsilon$-paths). Process $p$ sorts the maximal $\epsilon$-paths of $K_p$ (according to any predefined total order), and decides on the rank corresponding to its own $\epsilon$-path (necessarily the longest $\epsilon$-path in $W_p$).

**Algorithm 2.** TORCHE- process $p$

```
1  initial knowledge:
2      the network size n;
3  variables:
4      W_p : acyclic graph, reduced estimate of causal past, initially set to a single
       vertex (0, ⊥)
5  for round r = 1, 2, ...
6      (W_q)_{q∈N_p} ← scan()
7      b ← rand()                          /* possibly poor quality random bits */
8      W_p ← fold( (⊔_{q∈N_p∪{p}} W_q) ⊕ (r, b) )
9      let c be the number of maximal ε-paths in W_p
10     if c = n then
11         let K_p be the shortest prefix of W_p with n maximal ε-paths
12         sort the maximal ε-paths of K_p
13         decide on the rank of the ε-path corresponding to self
```

**Lemma 3.** *Let $P$ be a past cone at $p$. Then, at the end of $P$, there is an isomorphism $W_p \simeq \mathtt{fold}(P)$.*

*Proof.* In this proof, we denote by $W_p^r$ the value of the variable $W_p$ at the beginning of round $r$. We prove the claim by induction on the height $h$ of $P$. If $h = 0$, i.e., $P$ is reduced to a single vertex $(p, 0)$ labeled with $\perp$, then, since $W_p$ is initialized to a single vertex labeled with $\perp$, the folded graph $\mathtt{fold}(W_p^0)$

equals $W_p^0$ and the claim holds. Assume the result holds for all causal pasts of height at most $h$. Let $P$ be a causal past at $p$ of height $h+1$. Then, $P$ can be written as $P = \left(\bigcup_{q \in \mathcal{N}_p \cup \{p\}} J_q\right) \oplus (p, r+1)$, where $J_q$ is the maximal causal past of height at most $h$ at $q$ in $P$, and $r+1$ is the round number at $p$ at the end of $P$. For every $q \in \mathcal{N}_p \cup \{p\}$, let $r_q$ be the round number of process $q$ at the end of $J_q$. By the induction hypothesis, we have $\forall q \in \mathcal{N}_p \cup \{p\},\ W_q^{r_q} \simeq \texttt{fold}(J_q)$. Moreover, according to Algorithm 2 (line 8), we have

$$\begin{aligned} W_p^{r+1} &= \texttt{fold}\left(\left(\bigsqcup_{q \in \mathcal{N}_p \cup \{p\}} W_q^{r_q}\right) \oplus (r+1, b)\right) \\ &\simeq \texttt{fold}\left(\left(\bigsqcup_{q \in \mathcal{N}_p \cup \{p\}} \texttt{fold}(J_q)\right) \oplus (r+1, b)\right) \\ &\simeq \texttt{fold}(P). \end{aligned}$$

□

**Proposition 3.** *Given that the processes know the network size $n$, the* Torche *algorithm solves enumeration. Moreover, for every fair schedule $S$ with finite divergence time:* (i) *the height of the decision event at $p$ is at most $C_{cv}(C_{dv})[p]$,* (ii) *state and message size is $O(n^2 \cdot T^2)$ where $T = \max_p\{C_{cv}(C_{dv})[p]\}$.*

*Proof (Termination).* Let $P$ be a past cone at $p$ of height $C_{cv}(C_{dv})[p]$. By the definition of the cover time function, all processes participate in $P$. By definition, any two distinct events in $C_{dv}$ have non isomorphic causal pasts. Therefore, by Lemma 2, the number of maximal $\epsilon$-paths in $W = \texttt{fold}(P)$ is $n$. Thus $p$ decides at most at the end of $P$.

*(Uniqueness).* Let $p_1, p_2$ be two distinct processes. Let $P_1, P_2$ the causal pasts of their decision events respectively. Let $Z = [0, C_{dv}]$ be the prefix corresponding to the divergence cut. Necessarily, $Z$ is a prefix of $P_1$ and $P_2$. By Lemma 3, the values $K_1$ and $K_2$ computed at line 11 in Algorithm 2 both correspond to $\texttt{fold}(Z)$, thus $K_1 = K_2 \underset{def}{=} K$. In particular, processes $p_1$ and $p_2$ sort the maximal $\epsilon$-paths of $K$ the same way. And they decide on the ranks of two distinct $\epsilon$-paths.

*(State and message size).* When a process decides, the variable $W_p$ is (isomorphic to) the folding of a past cone of height at most $T$. Thus, it is possible to encode $W_p$ as an adjacency matrix of dimension $n \cdot T$ at most, which requires $O(n^2T^2)$ bits. □

## 5 Encompassing Previous Approaches

In this section, we explain how previous approaches can be understood in terms of divergence time. More precisely, we explain how the underlying assumptions

(lack of symmetry in the network topology, or use of independent random bits) imply that the divergence time is finite. Other conditions may be assumed. But were they sufficient to solve enumeration, they would necessarily imply that the divergence time is finite.

**(Fiber minimal networks).** As we pointed out, previous work [2,7–9,21–23] mainly focused on the spatial aspect: the topology of the network. We briefly recall the definition of a fibration. A *fibration* $\Psi : \mathcal{N} \to \mathcal{G}$ is a surjective graph morphism (preserving labels if any) such that, for any vertex $v$ in $\mathcal{G}$, for any vertex $q$ in $\Psi^{-1}(v)$ (the *fiber* over $v$), $\Psi$ induces a bijection (preserving labels if any) between the set of arrows into $v$ and the set of arrows into $q$. The network $\mathcal{N}$ is *fiber-minimal* if any fibration $\Psi : \mathcal{N} \to \mathcal{G}$ is an isomorphism. Refer to [7] for further details. The notion of fibration captures the "spatial similarities" among the vertices of a network. The following proposition relates the concept of fibration with that of divergence time.

For the sake of simplicity, we consider, in this section, networks with *port-awareness*, i.e., the processes are able to distinguish their incoming arrows. Formally, given $q \xrightarrow{\omega} p$ and $q' \xrightarrow{\omega'} p$, then $q = q'$ iff $\omega = \omega'$.

**Proposition 4.** *A network $\mathcal{N}$ with port-awareness is fiber-minimal if and only if all fair schedules have a finite divergence time.*

*Proof.* If the network $\mathcal{N}$ is not fiber-minimal, then [6,7] have shown how to design a schedule in which at least two processes are always indistinguishable, i.e., for any height $h$, their respective causal pasts of height $h$ are isomorphic. In particular, this schedule has an infinite divergence time. We proceed to prove the other direction: if there exists a schedule $S$ with infinite divergence time then the network is not fiber-minimal.

Let $S$ be a fair schedule with infinite divergence time on a network $\mathcal{N}$. Let $\Phi : S \to W = \texttt{fold}(S)$ be the surjective graph morphism as defined in Lemma 1. Let $k$ denote the number of infinite $\epsilon$-paths in $W$. By assumption, $k < n$. We define a network $\mathcal{G}$. The vertices of $\mathcal{G}$ are the infinite $\epsilon$-paths in $W$. For any two vertices $u, v$ in $\mathcal{G}$, there is an arrow $u \xrightarrow{\omega} v$ if and only if $u = u_1 \xrightarrow{\epsilon} u_2 \ldots$, $v = v_1 \xrightarrow{\epsilon} v_2 \ldots$, there exist $i, j \geq 1$ such that $u_i \xrightarrow{\omega} v_j$. Naturally, there is a surjective graph morphism $\Psi : \mathcal{N} \to \mathcal{G}$. This morphism maps a vertex $p$ in $\mathcal{N}$ to the infinite $\epsilon$-path in $W$ obtained as the image of the infinite $\epsilon$-path associated with $p$ in $S$ under the folding operation. We claim that $\Psi$ is a fibration. Indeed, let $q$ be a vertex in $\mathcal{N}$. Let $b = b_1 \xrightarrow{\epsilon} b_2 \ldots$ be the infinite $\epsilon$-path in $S$ corresponding to process $q$, and $v = \Phi(b) = \Psi(q) = v_1 \xrightarrow{\epsilon} v_2 \ldots$ the corresponding infinite $\epsilon$-path in $W$. Let $u \xrightarrow{\omega} v$ be a neighbour of $v$ in $\mathcal{G}$. We have to prove that there exists a unique arrow $p \xrightarrow{\omega} q$ in $\mathcal{N}$ such that $\psi(p) = u$.

We write $u = u_1 \xrightarrow{\epsilon} u_2 \ldots$. By definition, there exist $i, j \geq 1$ such that $u_i \xrightarrow{\omega} v_j$ in $W$. The existence of the arrow $u_i \xrightarrow{\omega} v_j$ implies that there exist vertices (events) $a'_i, b'_j$ in $S$ such that $\Phi(a'_i) = u_i$, $\Phi(b'_j) = v_j$ and $a'_i \xrightarrow{\omega} b'_j$. We also have $\Phi(b_j) = v_j$, that is, the events $b_j$ and $b'_j$ have isomorphic causal pasts. In particular, there exists an event $a_i$ in $S$ such that $\Phi(a_i) = u_i$ and $a_i \xrightarrow{\omega} b_j$.

Let $p$ be the process in $\mathcal{N}$ at which the event $a_i$ occurs. We have $p \xrightarrow{\omega} q$, and, by the fact that incoming edges have pairwise distinct labels, $p$ is the unique neighbour of $q$ with such an arrow. It remains to show that $\Psi(p) = u$. Then $p \xrightarrow{\omega} q$. Moreover, the infinite $\epsilon$-path $a = a_1 \xrightarrow{\epsilon} a_2 \dots$ in $S$ corresponding to $p$ is the one going through the event $a_i$. The infinite $\epsilon$-path $\Phi(a)$ in $W$ is necessarily the path $u$, because, by definition, $u_i$ is such that $a_i$ occurs in $[C_k, \infty)$, and thus there is a unique infinite $\epsilon$-path in $W$ going through $u_i$. In particular, $\Psi(p) = u$.

To conclude, we have shown that $\Psi : \mathcal{N} \to \mathcal{G}$ is a fibration. Since $\mathcal{G}$ has strictly less vertices ($k < n$), $\mathcal{N}$ is not fiber-minimal. □

**(Randomness).** We argue that the main purpose of using randomness is to have a finite divergence with high probability. For the sake of simplicity, we consider the case of synchronous bidirectional complete network (without arrow labels). At every round $r$, process $p$ reads a random bit $b_p^r$.

**Proposition 5.** *Assuming that $(b_p^r)_{p \in \mathcal{VN}, r \in \mathbb{N}}$ are mutually independent uniform random bits, we have $\tau_{dv} = \max_p\{C_{dv}[p]\} = O(\log n)$ with high probability.*

*Proof.* Fix an arbitrary integer $k$. Consider, for each process $p$, the sequence $w_p = (b_p^1, \dots, b_p^k)$ of the $k$ bits read during the first $k$ rounds. The problem of computing the probability $\alpha$ that $w_p = w_q$ for at least two distinct processes $p, q$ is an instance of the birthday paradox ($n$ people with $2^k$ possible birthday dates). This yields $\alpha \simeq 1 - e^{-\frac{n^2}{2^{k+1}}}$. Setting $k = O(\log n)$, we obtain $\alpha \simeq 1 - e^{-O(n)}$. □

## 6 Related Work

Our computational model is the classical asynchronous message-passing model [14]. Our formulation of this model can be seen as the asynchronous generalization of the $\mathcal{LOCAL}$ model of Linial [13]. Many approaches have addressed the issue of symmetry breaking in the $\mathcal{LOCAL}$ model. Leveraging the synchronous nature of $\mathcal{LOCAL}$, these approaches have mainly focused on computing the time complexity of problems like maximum independent set, maximal matching, coloring or network decomposition [3,5,10,13,17–19]. In most cases, these approaches assume processes with identifiers [3,17]. Our paper addresses the question of symmetry breaking in the more general settings of asynchronous computation in purely anonymous networks (no identifiers). In particular, we address the fundamental problem of enumeration on arbitrary networks. As explained in the introduction, the main obstacle to symmetry breaking problems in anonymous networks has been formulated by Angluin in [2], under the form of graph coverings, a concept borrowed from algebraic topology [15]. Later on, Yamashita and Kameda [21–23] extended Angluin's work, and inspired by the work of Johnson and Schneider [12], introduced the notion of *view* of a process, to encode all the information accessible to a process as the rooted tree of the finite labeled walks in the network from that process. In [20], the author presents a method to compress a view, thereby enhancing the space and time complexity of some

symmetry breaking algorithms. In [6], Boldi et al. considered several synchronous and asynchronous models, and provided a characterization based on the notion of *graph fibration* [7], another concept borrowed from algebraic topology, which refines that of graph covering. In [8,9], Chalopin et al., inspired by the work of Mazurkiewicz [16], studied symmetry breaking in message-passing, and presented message-efficient algorithms in the context of networks without spatial symmetries.

**Acknowledgment.** This work has been supported in part by the European ERC Grant 339539 - AOC.

## References

1. Afek, Y., Matias, Y.: Elections in anonymous networks. Inf. Comput. **113**(2), 312–330 (1994)
2. Angluin, D.: Local and global properties in networks of processors. In: 12th Symposium on the Theory of Computing, pp. 82–93. ACM (1980)
3. Awerbuch, B., Goldberg, A.V., Luby, M., Plotkin, S.A.: Network decomposition and locality in distributed computation. In: 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October–1 November 1989, pp. 364–369 (1989)
4. Babai, L.: Graph isomorphism in quasipolynomial time [extended abstract]. In: Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, 18–21 June 2016, pp. 684–697 (2016)
5. Barenboim, L., Elkin, M., Kuhn, F.: Distributed $(\Delta + 1)$-coloring in linear (in $\Delta$) time. SIAM J. Comput. **43**(1), 72–95 (2014)
6. Boldi, P., Shammah, S., Vigna, S., Codenotti, B., Gemmell, P., Simon, J.: Symmetry breaking in anonymous networks: characterizations. In: Israel Symposium on Theory of Computing and Systems, pp. 16–26 (1996)
7. Boldi, P., Vigna, S.: Fibrations of graphs. Discrete Math. **243**(1–3), 21–66 (2002)
8. Chalopin, J., Métivier, Y.: An efficient message passing election algorithm based on Mazurkiewicz's algorithm. Fundamenta Informaticae **80**(1–3), 221–246 (2007)
9. Chalopin, J., Métivier, Y., Morsellino, T.: Enumeration and leader election in partially anonymous and multi-hop broadcast networks. Fundamenta Informaticae **120**(1), 1–27 (2012)
10. Hanckowiak, M., Karonski, M., Panconesi, A.: On the distributed complexity of computing maximal matchings. SIAM J. Discrete Math. **15**(1), 41–57 (2001)
11. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. In: 22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28–30 October 1981, pp. 150–158 (1981)
12. Johnson, R.E., Schneider, F.B.: Symmetry and similarity in distributed systems. In: Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC 1985, pp. 13–22. ACM, New York (1985)
13. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
14. Lynch, N.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., Burlington (1997)
15. Massey, W.S.: A Basic Course in Algebraic Topology. Springer, New York (1991)

16. Mazurkiewicz, A.: Distributed enumeration. Inf. Process. Lett. **61**(5), 233–239 (1997)
17. Panconesi, A., Srinivasan, A.: On the complexity of distributed network decomposition. J. Algorithms **20**(2), 356–374 (1996)
18. Schneider, J., Wattenhofer, R.: An optimal maximal independent set algorithm for bounded-independence graphs. Distrib. Comput. **22**(5–6), 349–361 (2010)
19. Szegedy, M., Vishwanathan, S.: Locality based graph coloring. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, San Diego, CA, USA, 16–18 May 1993, pp. 201–207 (1993)
20. Tani, S.: Compression of view on anonymous networks - folded view -. IEEE Trans. Parallel Distrib. Syst. **23**(2), 255–262 (2012)
21. Yamashita, M., Kameda, T.: Computing on anonymous networks: part I - characterizing the solvable cases. IEEE Trans. Parallel Distrib. Syst. **7**(1), 69–89 (1996)
22. Yamashita, M., Kameda, T.: Computing on anonymous networks: part II - decision and membership problems. IEEE Trans. Parallel Distrib. Syst. **7**(1), 90–96 (1996)
23. Yamashita, M., Kameda, T.: Leader election problem on networks in which processor identity numbers are not distinct. IEEE Trans. Parallel Distrib. Syst. **10**(9), 878–887 (1999)

# Mobile Agents

# A General Lower Bound for Collaborative Tree Exploration

Yann Disser[1(✉)], Frank Mousset[2], Andreas Noever[2], Nemanja Škorić[2], and Angelika Steger[2]

[1] Institute of Mathematics, Graduate School CE, TU Darmstadt, Darmstadt, Germany
disser@mathematik.tu-darmstadt.de
[2] Department of Computer Science, ETH Zurich, Zurich, Switzerland
{moussetf,anoever,nskoric,steger}@inf.ethz.ch

**Abstract.** We consider collaborative graph exploration with a set of $k$ agents. All agents start at a common vertex of an initially unknown graph with $n$ vertices and need to collectively visit all other vertices. We assume agents are deterministic, moves are simultaneous, and we allow agents to communicate globally. For this setting, we give the first non-trivial lower bounds that bridge the gap between small ($k \leq \sqrt{n}$) and large ($k \geq n$) teams of agents. Remarkably, our bounds tightly connect to existing results in both domains.

First, we significantly extend a lower bound of $\Omega(\log k / \log\log k)$ by Dynia et al. on the competitive ratio of a collaborative tree exploration strategy to the range $k \leq n \log^c n$ for any $c \in \mathbb{N}$. Second, we provide a tight lower bound on the number of agents needed for any competitive exploration algorithm. In particular, we show that any collaborative tree exploration algorithm with $k = Dn^{1+o(1)}$ agents has a competitive ratio of $\omega(1)$, while Dereniowski et al. gave an algorithm with $k = Dn^{1+\varepsilon}$ agents and competitive ratio $\mathcal{O}(1)$, for any $\varepsilon > 0$ and with $D$ denoting the diameter of the graph. Lastly, we show that, for any exploration algorithm using $k = n$ agents, there exist trees of arbitrarily large height $D$ that require $\Omega(D^2)$ rounds, and we provide a simple algorithm that matches this bound for all trees.

## 1 Introduction

Graph exploration captures the problem of navigating an unknown terrain with a single or multiple autonomous robots. In the abstract setting, we take the perspective of an agent that is located at some vertex of an initially unknown

Y. Disser—Supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School CE at TU Darmstadt.

F. Mousset—Supported by grant no. 6910960 of the Fonds National de la Recherche, Luxembourg.

A. Noever—Supported by grant no. 200021 143338 of the Swiss National Science Foundation.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 125–139, 2017.
https://doi.org/10.1007/978-3-319-72050-0_8

graph, can locally distinguish edges at its current location, and can choose an edge to traverse in its next move. Various scenarios for graph exploration have been studied in the past, for different graph classes and different capabilities of the agent(s). A fundamental goal of exploration is to systematically visit all vertices/edges of the underlying graph. For settings where exploration is possible, we typically ask for efficient exploration algorithms, e.g., in terms of the number of edge traversals.

In this paper, we consider *collaborative* exploration, where a set of $k$ agents are initially located at some vertex of an unknown *undirected* graph. We assume agents to move deterministically, allow them to freely communicate at all times, and to have unlimited computational power and memory at their disposal. In every round each agent may traverse any edge incident to its current location, where the edges incident to a vertex are revealed when that vertex is visited for the first time. The goal is to visit all vertices while minimizing the number of rounds. More precisely, we are interested in the competitive ratio of an exploration strategy, i.e., the worst case ratio between the total number of rounds it needs and the minimum total number of rounds needed to visit all vertices of the same graph, assuming it is known beforehand. We prove new lower bounds for the best-possible competitive ratio of any collaborative exploration algorithm. Our bounds hold even for the much simpler setting of *tree* exploration. Note that since our results concern trees, it makes no difference whether nodes can be distinguished, and whether the agents need to visit all edges or not.

Let $\mathcal{T}_{n,D}$ denote set of all rooted trees with $n$ vertices and height $D$. Each such tree corresponds to an instance of the tree exploration problem in which all $k$ agents start at the root of the tree. Clearly, any offline exploration algorithm needs $\Omega(n/k+D)$ rounds to explore a tree in $\mathcal{T}_{n,D}$ using $k$ agents. This is shown to be tight by the following offline exploration algorithm that explores the tree in $\Theta(n/k+D)$ rounds: start with the tree $T$, double its edges, find an Eulerian tour $C$ (of length $2n-2$), distribute the agents evenly on $C$ (this takes at most $D$ rounds), and explore $T$ by letting each agent walk along $C$ for $\mathcal{O}(n/k)$ rounds.

In the online setting, we can explore a tree in $\mathcal{T}_{n,D}$ with a single agent using a depth-first traversal in time $\mathcal{O}(n)$ and thus we trivially have a competitive ratio of $\mathcal{O}(1)$ when $k$ is constant. On the other hand, with $k \geq \Delta^D$ agents, where $\Delta$ is the maximum degree of the tree, we can simply perform a breadth-first traversal, which takes $\mathcal{O}(D)$ steps and thus also has competitive ratio $\mathcal{O}(1)$. Observe that in the first case $n/k$ dominates the lower bound on the offline optimum, while in the second case $D$ is dominating. We are interested in the best-possible competitive ratios between these two extreme cases.

Surprisingly, Dereniowski et al. [12] showed that already a polynomial number $k = Dn^{1+\varepsilon}$ of agents allows for a BFS-like algorithm that achieves a constant competitive ratio. For smaller teams of agents, Fraigniaud et al. [18,20] gave a collaborative algorithm with competitive ratio $\mathcal{O}(k/\log k)$. This is only slightly better than the trivial upper bound of $\mathcal{O}(k)$ that we get by performing a depth first traversal with a single agent. Ortolf and Schindelhauer [23] improved this competitive ratio to $k^{o(1)}$ for $k = 2^{\omega(\sqrt{\log D \log\log D})}$ and $n = 2^{\mathcal{O}(2^{\sqrt{\log D}})}$. The

only non-trivial lower bound for collaborative tree exploration was given by Dynia et al. [16]. They showed that any deterministic exploration algorithm for $k < \sqrt{n}$ agents has competitive ratio $\Omega(\log k / \log\log k)$.

### Our Results

We give the first non-trivial lower bounds on the competitive ratio for collaborative tree exploration in the domain $k \geq \sqrt{n}$ (cf. Fig. 1). More precisely, we show that for every constant $c \in \mathbb{N}$, any given deterministic exploration strategy with $k \leq n \log^c n$ agents has competitive ratio $\Omega(\log k / \log\log k)$ on the set of all trees on $n$ vertices. Note that this extends the range of the bound by Dynia et al. [16] for $k < \sqrt{n}$ significantly.

Secondly, we show that for every constant $\varepsilon > 0$, there is a constant $D = D(\varepsilon)$ such that for any exploration algorithm with $k \leq Dn^{1+\varepsilon}$ agents, there exists a tree in $\mathcal{T}_{n,D}$ on which the algorithm needs at least $D/(5\varepsilon)$ rounds. This (almost) tightly matches the algorithm of Dereniowski et al. [12], which can explore any tree in at most $(1 + o(1))D/\varepsilon$ rounds using $k = Dn^{1+\varepsilon}$ agents. Our result implies that any exploration algorithm with $k = Dn^{1+o(1)}$ agents has competitive ratio $\omega(1)$. More precisely, we get that for any function $0 \leq f(n) \leq o(1)$, there is a function $D = D(n)$ such that every exploration algorithm with $k = Dn^{1+f(n)}$ agents has competitive ratio $\omega(1)$ on the trees in $\mathcal{T}_{n,D}$. In contrast, the algorithm of Dereniowski et al. shows that $k = Dn^{1+\varepsilon}$ agents are sufficient to get a competitive ratio $\mathcal{O}(1)$ on such trees.

Finally, for every exploration algorithm with $k = n$, we construct a tree of height $D = \omega(1)$ where the algorithm needs $\mathcal{O}(D^2)$ rounds. We give a simple algorithm that achieves this bound in general.

### Further Related Work

Many variants of graph exploration with a *single agent* have been studied in the past. Any (strongly) connected graph with *distinguishable* vertices can easily be explored in polynomial time by systematically building a map of the graph. Regarding the exploration of *undirected* graphs with *indistinguishable* vertices, Aleliunas et al. [2] showed that a random walk explores any graph in $\mathcal{O}(n^3\Delta^2 \log n)$ steps, with high probability. In order to turn this into a terminating exploration algorithm the agent needs $\Omega(\log n)$ bits of memory. Fraigniaud et al. [19] showed that every deterministic algorithm needs $\Omega(\log n)$ bits of memory, and Reingold [25] gave a matching upper bound. Disser et al. [14] showed that alternatively $\Theta(\log\log n)$ pebbles and bits of memory are necessary and sufficient for exploration, where a pebble is a device that can be dropped to make a vertex distinguishable and that can be picked up and reused later. Diks et al. [13] showed that trees can be explored with $\mathcal{O}(\log \Delta)$ memory, and that $\Omega(\log n)$ memory is required if the agent needs to eventually terminate at the start vertex. Ambühl et al. [4] gave a matching upper bound for the latter result.

For the case of *directed* graphs with *distinguishable* vertices, Albers and Henzinger [1] gave an exploration algorithm with subexponential running time

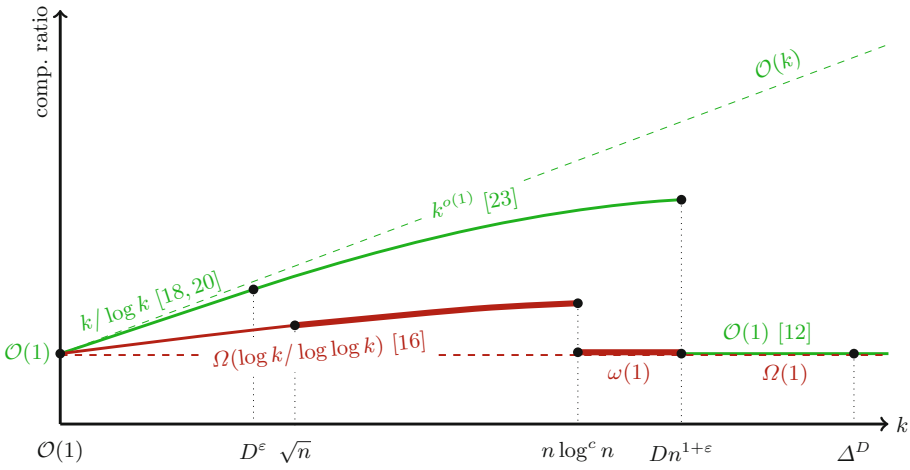



**Fig. 1.** State of the art in collaborative tree exploration. The top curve shows upper and the bottom curve lower bounds. Thick lines show our results.

$d^{\mathcal{O}(\log d)}m$ that learns a map of the graph. Here $m$ denotes the number of edges and $d$ is the deficiency of the graph, i.e., the number of edges missing to make the graph Eulerian. This results narrows the gap between a quadratic lower bound and an exponential upper bound introduced by Deng and Papadimitriou [11].

An even more challenging setting (for the agent) is the exploration of *directed*, strongly connected graphs with *indistinguishable* vertices. In general the agent needs exponential time to explore a graph in this setting. On the other hand, Bender and Slonim [7] showed that two agents can explore any directed graph in polynomial time, using a randomized strategy. Bender et al. [6] showed that to accomplish this with a single agent we need $\Theta(\log\log n)$ pebbles, i.e., "a friend is worth $\mathcal{O}(\log\log n)$ pebbles". Remarkably, Bender et al. [6] also showed that if the number of vertices is known beforehand, a deterministic agent with a single pebble can explore any directed graph in polynomial time $\mathcal{O}(n^8\Delta^2)$.

The lower bounds for collaborative tree exploration discussed above carry over to the *collaborative exploration* of general undirected graphs with distinguishable vertices. Also, the algorithm of Dereniowski et al. [12] for $k = Dn^{1+\varepsilon}$ works on general graphs. Additionally, Ortolf and Schindelhauer [22] gave a lower bound on the best-possible competitive ratio for randomized algorithms of $\Omega(\sqrt{\log k}/\log\log k)$ for $k = \sqrt{n}$. Collaborative exploration by multiple random walks without communication has been considered by Alon et al. [3], Elsässer and Sauerwald [17], and Ortolf and Schindelhauer [24].

Graph exploration has been studied in many other settings. Examples include tethered exploration or exploration with limited fuel [5,15], exploration of mazes [8,21], and exploration of polygonal environments [9,10].

## 2 Results

Our first result extends the lower bound for $k < \sqrt{n}$ agents of Dynia et al. [16] to the much larger range $k \leq n \log^{\mathcal{O}(1)} n$. We prove the following theorem:

**Theorem 1.** *Let $c$ be any positive integer constant. Then for every $n$ and every $1 \leq k \leq n \log^c n$ there is some $D = D(n, k, c)$ such that the following holds: for any given deterministic exploration strategy with $k$ agents, there exists a tree $T$ on $n$ vertices and with height $D$ on which the strategy needs*

$$\Omega\Big( \frac{\log k}{\log \log k} \cdot (n/k + D) \Big)$$

*rounds.*

As mentioned above, there is an offline algorithm that explores any graph with $n$ vertices and height $D$ in time $\Theta(n/k + D)$. From this, we obtain the following corollary to Theorem 1:

**Corollary 1.** *Let $c$ be any positive integer constant. Then any deterministic exploration strategy using $k \leq n \log^c n$ agents has a competitive ratio of*

$$\Omega\Big( \frac{\log k}{\log \log k} \Big).$$

Our second main result shows that the algorithm of Dereniowski et al. [12] that explores a graph with $k = Dn^{1+\varepsilon}$ agents in time $(1 + o(1))D/\varepsilon$ is almost optimal: using $k \leq Dn^{1+\varepsilon}$ agents it is generally impossible to explore the graph in fewer than $D/(5\varepsilon)$ rounds.

**Theorem 2.** *Given any constant $\varepsilon > 0$ there is an integer $D = D(\varepsilon)$ such that for sufficiently large $n$ and for every deterministic exploration strategy using $k \leq D \cdot n^{1+\varepsilon}$ agents, there exists a tree on $n$ vertices and with height $D$ on which the strategy needs at least $D/(5\varepsilon)$ rounds.*

In the range where $k \geq n$, the offline optimum is determined by the height $D$ of the tree. Therefore, the result of Dereniowski et al. mentioned above implies that the competitive ratio is constant when $k = D \cdot n^{1+\Omega(1)}$. Theorem 2 shows in particular that this is tight in the following sense:

**Corollary 2.** *For any function $0 \leq f(n) \leq o(1)$, there is a function $D = D(n)$ such that the competitive ratio of any deterministic exploration strategy using $k = D \cdot n^{1+f(n)}$ agents is $\omega(1)$ on the set $\mathcal{T}_{n,D}$ of all rooted trees with $n$ vertices and height $D$.*

However, note that here we have no control over the height of the worst-case example: for instance, it could be that there are ranges for $D$ where the algorithm of Dereniowski et al. may be improved.

Finally, it is possible for $k = n$ agents to explore any tree on $n$ vertices and of height $D$ in $D^2$ rounds using a breadth-first exploration strategy. More precisely, we can split the $D^2$ rounds in $D$ phases of length $D$, and in each phase $1 \leq i \leq D$ do the following. Let $A_i$ be the set of unvisited leaves of the tree that is known to the agents at the start of phase $i$. Then we send one agent to each vertex in $A_i$ along a shortest path. This is clearly doable in $D$ rounds, and constitutes a single phase. After phase $i$, the agents have explored all vertices at distance at most $i$ from the root. Therefore, after $D$ such phases, the tree is completely explored. We show that the running time of $D^2$ is optimal up to a constant factor:

**Theorem 3.** *For every $n$ and every deterministic exploration strategy using $k = n$ agents, there exists a tree $T$ on $n$ vertices and with height $D = \omega(1)$ such that the strategy needs at least $D^2/3$ rounds to explore $T$.*

In all the results above, we have considered the worst-case performance of an exploration strategy on any tree. However, by looking at the proofs of Theorems 1 and 2, one can see that the heights of our lower bound constructions are typically quite small. We believe it is also natural to ask about the competitive ratio on the set of trees of height at least $D$, for a given $D$. We show that at least for subpolynomial heights, the competitive ratio with $k = \Theta(n)$ agents is unbounded:

**Theorem 4.** *For any function $D \leq n^{o(1)}$ and any exploration strategy using $k = \Theta(n)$ agents, the competitive ratio on the set of all trees of size $n$ and height at least $D$ is $\omega(1)$.*

We stress that Theorem 4 differs from the other results in that it gives a measure of control over the height of the adversarial example, while the other results merely state that there *exists* some height on which the algorithm must perform poorly.

## 3 Tree Exploration Games

In order to prove a lower bound on the competitive ratio, we consider a tree exploration game defined as follows. By a *tree exploration game with $k$ agents* we mean a game with two players, the *explorer* (the online algorithm) and the *revealer* (the adversary), played according to the following rules. The game proceeds in rounds which we index by the variable $t$ ('time'), the first round being $t = 0$. The state of the game at time $t$ is described by a triple $(T_t, A_t, \phi_t)$, where $T_t$ is a rooted tree (the tree revealed at the beginning of round $t$), $A_t$ is a subset of the vertices of $T_t$ (the subset of *visited* vertices by round $t$), and $\phi_t \colon \{1, \ldots, k\} \to A_t$ is an assignment of the agents to the vertices (where $\phi_t(i)$ is the location of the $i$-th agent at time $t$). In round $t = 0$ the revealer decides on the initial tree $T_0$. The state at time 0 is then given by $(T_0, A_0, \phi_0)$ where $A_0 = \{\text{root}(T_0)\}$ and $\phi_0(x) = \text{root}(T_0)$ for all $1 \leq x \leq k$ – that is to say, all

agents are initially at the root of $T_0$. In every round $t > 0$, each player can make a move. First, the explorer creates a new assignment $\phi_t$ by moving each agent $i$ to a neighbor of $\phi_{t-1}(i)$ in $T_{t-1}$ or by keeping the location of the agent same, i.e., $\phi_t(i) = \phi_{t-1}(i)$. Then the revealer decides on the new tree $T_t$, where $T_t$ must be obtained from $T_{t-1}$ by attaching (possibly empty) trees at some vertices $v \in V(T_{t-1}) \setminus A_{t-1}$, where $V(T_{t-1})$ is the set of vertices of $T_{t-1}$. We then let $A_t = A_{t-1} \cup N_t$ where $N_t = \{\phi_t(i) : 1 \leq i \leq k\}$ is the set of the new agent locations. The game ends in round $t^*$ if all vertices of $T_{t^*}$ are visited at the beginning of round $t^*$, i.e., if $A_{t^*} = V(T_{t^*})$.

This type of game naturally lends itself to proving lower bounds for the time in which $k$ agents can explore an unknown tree. Specifically, consider any deterministic strategy for exploring an unknown tree $T$ with $k$ agents. Such a strategy can be interpreted as a strategy for the explorer in the tree exploration game with $k$ agents. If the revealer can play so that the game lasts for at least $t^*$ rounds, then this means that the proposed exploration strategy needs $t^*$ rounds to explore the tree $T_{t^*}$. We will use this observation to prove lower bounds for the online graph exploration in the following section.

As a side remark, here it is crucial that the strategy is deterministic: if the strategy were allowed to make random choices, then the tree $T_{t^*}$ would turn out to be a random variable that might be highly correlated with the random choices made by the explorer, and it could not serve as an instance on which the strategy performs badly.

## 4 Lower Bound Construction

We now give our lower bound construction that establishes the following technical lemma.

**Lemma 1.** *Let $n, L, m$ be positive integers such that $n \geq L \cdot 16^m$. Then for any deterministic exploration strategy using*

$$k \leq \frac{n^{1+1/m}}{6L(m+1)^2(2L)^{1/m}}$$

*agents, there exists a tree $T$ on $n$ vertices and of height $Lm$ such that the strategy needs at least $L\binom{m}{2}$ rounds to explore $T$.*

The parameter $L$ is mostly there to force a large diameter. For a first understanding it does not hurt to imagine that $L = 1$ and to think of $m$ as being a function tending to infinity very slowly as $n$ grows. Then the lemma shows that $n^{1+o(1)}$ vertices need $\omega(1)$ rounds to explore the tree.

*Proof.* Assume that integers $n$, $L$ and $m$ as above are given. Let $k$ be any integer such that $1 \leq k \leq n^{1+1/m}/(6L(m+1)^2(2L)^{1/m})$. To prove the lemma, we will describe a strategy for the revealer in the tree exploration game with $k$ agents such that

– the game does not end before round $t^* := L \cdot \binom{m}{2}$, and
– the tree $T_{t^*}$ has height $Lm$ and at most $n$ vertices,

where the notation is as in Sect. 3. Note that this is enough to prove the lemma.

The main difficulty is that there are several trade-offs involved. On the one hand, the game has to keep going for $t^*$ rounds, that is, it must not happen that the agents explore the whole tree at any time before $t^*$. This requires us to grow the tree at several critical times, when the agents may come close to exploring everything. On the other hand, we do not want to grow the tree too often, or too much, because the final tree must consist of at most $n$ vertices. Lastly, there is the (less severe) constraint that we want the constructed tree to have a certain height, which we must keep in mind.

Before explaining the strategy, we fix some notation. Let

$$\alpha := (2L/n)^{1/m} \quad \text{and} \quad t_i := L \cdot \binom{i+1}{2}$$

for $0 \leq i < m$. For each $t \geq 0$ we can consider the equivalence relation $\sim_t$ on $V(T_t)$ where $u \sim_t v$ if there exists a path between $u$ and $v$ in $T_t$ that avoids the root of $T_t$ (i.e., if they have a common ancestor that is not the root). Since $T_0 \subseteq T_1 \subseteq T_2 \subseteq \ldots$ are trees with the same root, we will just write $u \sim v$ instead of $u \sim_t v$ without causing confusion. Then we define $a_t(v) := |\{x \mid \phi_t(x) \sim v\}|$. In other words, $a_t(v)$ counts the total number of agents that could reach vertex $v$ without passing through the root (under the assignment $\phi_t$). We think of those agents as being 'near' the vertex $v$.

The times $t_1, t_2, t_3, \ldots$ are our 'critical times' at which the tree grows. The general idea is very natural: at every critical time $t_i$, we grow the tree in those places where there are the fewest agents nearby. When doing this, we add sufficiently many vertices so that the agents that are currently nearby cannot explore the newly added subtrees in before the next critical time $t_{i+1}$. Similarly, everything is set up so that the agents that are not nearby are unable to reach the location before the time $t_{i+1}$. Thus, the game keeps going until round $t_{i+1}$. The parameter $\alpha$ enforces a sort of 'iterative thinning' of the tree, which allows us to be economical with the vertices. A precise description of the revealing strategy is given in Algorithm 1.

The set $S_i$ is the set of vertices where we grow the tree at time $t_i$. For a better intuition, we refer the reader to Fig. 2, which shows what the tree constructed by this strategy might look like. We establish three claims which are used to show that the algorithm indeed runs for at least $t^* = t_{m-1}$ rounds and that the tree constructed in this way has the right properties.

*Claim 1.* For every $0 \leq i < m$ the following holds. The height of $T_{t_i}$ is at most $L \cdot (i+1)$. Moreover, if $S_1, \ldots, S_i$ are all non-empty, then the height of $T_{t_i}$ is exactly $L \cdot (i+1)$.

*Proof.* The tree $T_{t_i}$ differs from $T_{t_{i-1}}$ if and only if $S_i$ is non-empty, and in this case it is obtained by attaching trees of height $L$ at some vertices with distance

**Algorithm 1.** The strategy for the revealer.

**begin**
  let $T_0$ be a 'star' consisting of $\lceil n/(2L) \rceil$ paths of length $L$ from the root;
  **foreach** *round* $t = 1, 2, 3, \ldots$ **do**
    let the explorer choose $\phi_t$;
    **if** $t = t_i$ *for some* $1 \leq i < m$ **then**
      let $K_i$ be a maximal set of vertices in $V(T_{t_i-1}) \setminus A_{t_i-1}$ s.t.

      (i) every vertex in $K_i$ has distance $L \cdot i$ to the root in $T_{t_i-1}$
      (ii) there are no two distinct vertices $u, v \in K_i$ with $u \sim v$.

      let $S_i \subseteq K_i$ be the $\lceil \alpha |K_i| \rceil$ vertices $v \in K_i$ with least $a_{t_i}(v)$;
      define $T_{t_i}$ by attaching at each $v \in S_i$ a path of length $L-1$ with a star with $L \cdot (i+1) \cdot a_{t_i}(v)$ leaves at the end;
    **else**
      let $T_t = T_{t-1}$;
    **end**
  **end**
**end**

$L \cdot i$ to the root in $T_{t_{i-1}}$. Since $T_{t_0} = T_0$ has height $L$, this implies the claim by induction. □

*Claim 2.* For all $1 \leq i < m$ and every $v \in S_i$, there exists at least one descendant of $v$ at depth $L \cdot (i+1)$ in $T_{t_{i+1}-1}$ that does not belong to $A_{t_{i+1}-1}$. In particular, for all $1 \leq i < m$ we have $|K_{i+1}| = |S_i|$.

*Proof.* Each vertex at depth $L(i+1)$ is a descendant of some vertex $v \in S_i$. Moreover, we have $u \nsim v$ for any two distinct $u, v \in S_i$. Thus, the second claim follows directly from the first.

For the first claim, consider any $1 \leq i < m$ and $v \in S_i$. Note that

(1) at time $t_i$ we create $L \cdot (i+1) \cdot a_{t_i}(v)$ descendants of $v$ at depth $L \cdot (i+1)$;
(2) $t_{i+1} - t_i = L \cdot (i+1)$.

Because of this, no agent passing through the root can visit any descendant of $v$ at depth $L \cdot (i+1)$ before round $t_{i+1}$. On the other hand, the $a_{t_i}(v)$ agents that could visit a descendant at this depth without passing through the root cannot visit *all* descendants before round $t_{i+1}$. Thus at least one descendant at depth $L \cdot (i+1)$ must be unvisited at the end of round $t_{i+1} - 1$. □

*Claim 3.* For every $1 \leq i < m$ we have the bounds

$$|S_i| \geq \frac{\alpha^i n}{2L} \geq \frac{1}{\alpha} \quad \text{and} \quad |S_i| \leq \frac{(2\alpha)^i n}{2L}.$$

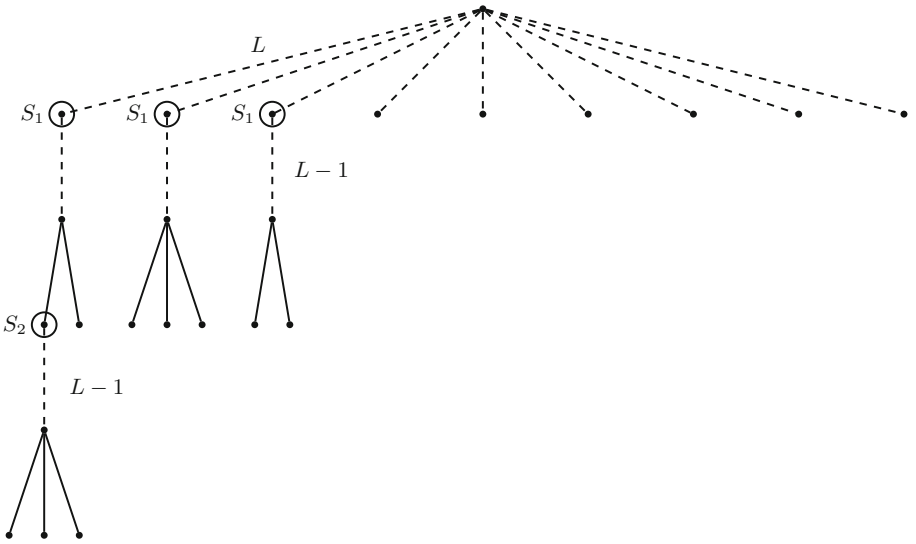



**Fig. 2.** A sketch of the tree generated by the revealing strategy, for artificial values $\alpha = 1/3$ and $\lceil n/(2L) \rceil = 9$ (degrees in the actual construction are much larger). The actual shape depends on the distribution of the agents at times $t_1, t_2$. Dashed lines represent paths of the specified length.

*Proof.* By definition we have $\alpha = (2L/n)^{1/m} < 1$ and thus $\alpha^m = 2L/n$, which gives us

$$\frac{\alpha^i n}{2L} \geq \frac{\alpha^{m-1} n}{2L} = 1/\alpha$$

for all $1 \leq i < m$.

For the lower bound, note that since $A_0$ contains only the root, we have $|K_1| = \lceil n/(2L) \rceil$. By the definition of $S_i$, we have $|S_i| \geq \alpha |K_i|$ for all $1 \leq i < m$. Moreover, if $2 \leq i < m$ then by Claim 2 we have $|K_i| = |S_{i-1}|$. The lower bound then follows by induction.

For the upper bound, note that $K_1 \leq n/(2L) + 1 \leq n/L$, where the last inequality uses $n \geq 2L$. Moreover, using $|K_i| \geq 1/\alpha$ we have $|S_i| \leq \alpha |K_i| + 1 \leq 2\alpha |K_i|$ for all $1 \leq i < m$. Finally, if $2 \leq i < m$ then $|K_i| = |S_{i-1}|$ by Claim 2, and the upper bound follows by induction. □

Since $|S_i| > 0$ implies in particular that $A_{t_i - 1} \neq V(T_{t_i - 1})$, we conclude from Claim 3 that the game does not stop before reaching round $t_{m-1} = L \cdot \binom{m}{2} = t^*$. Moreover, from Claims 1 and 3 we see that $T_{t_{m-1}}$ is a tree with height $L \cdot m$. To complete the proof we need to show that $|V(T_{t_{m-1}})| \leq n$. We have

$$|V(T_{t_{m-1}})| \leq \lceil n/(2L) \rceil \cdot L + 1 + \sum_{i=1}^{m-1} \sum_{v \in S_i} (L - 1 + L \cdot (i+1) \cdot a_{t_i}(v))$$

$$\leq n/2 + L + 1 + \sum_{i=1}^{m-1} L(i+1) \sum_{v \in S_i} a_{t_i}(v) + \sum_{i=1}^{m-1} |S_i|(L-1). \quad (1)$$

To bound the double sum note that $|K_i| \geq |S_i| \geq 1/\alpha$ (Claim 3) implies that $\lceil \alpha |K_i| \rceil \leq 2\alpha |K_i|$. Note also that the sum $\sum_{v \in K_i} a_{t_i}(v)$ in (1) is at most $k$, as no two vertices $u, v$ from $K_i$ are in the same subtree, i.e., $u \not\sim v$. Since $S_i$ contains the $\lceil \alpha |K_i| \rceil \leq 2\alpha |K_i|$ vertices of $K_i$ with least $a_{t_i}(v)$, we thus have

$$\sum_{v \in S_i} a_{t_i}(v) \leq 2\alpha \sum_{v \in K_i} a_{t_i}(v) \leq 2\alpha k,$$

and therefore

$$\sum_{i=1}^{m-1} L(i+1) \sum_{v \in S_i} a_{t_i}(v) \leq L(m+1)^2 \alpha k. \quad (2)$$

To bound the simple sum in (1), we use the upper bound from Claim 3 and obtain

$$\sum_{i=1}^{m-1} |S_i|(L-1) \leq (L-1) \sum_{i=1}^{\infty} \frac{(2\alpha)^i n}{2L} = \frac{L-1}{2L} \cdot 2\alpha n \sum_{i=0}^{\infty} (2\alpha)^i \leq \frac{2\alpha n}{2 - 4\alpha}. \quad (3)$$

Combining (1) with (2) and (3), we get

$$|V(T_{t_{m-1}})| \leq n/2 + L + 1 + L(m+1)^2 \alpha k + \frac{2\alpha n}{2 - 4\alpha}. \quad (4)$$

Since $n \geq L \cdot 16^m \geq 12L$ we have $L + 1 \leq 2L \leq n/6$. By the definition $\alpha = (2L/n)^{1/m}$ and the assumption $k \leq n^{1+1/m}/(6L(m+1)^2(2L)^{1/m})$ we have

$$L(m+1)^2 \alpha k = L(m+1)^2 (2L/n)^{1/m} k \leq n/6.$$

Finally, $n \geq L \cdot 16^m$ implies that $\alpha \leq 1/8$ and so the last term in (4) is also at most $n/6$. Hence $|V(T_{t_{m-1}})| \leq n/2 + 3n/6 = n$. □

## 5 Consequences for Competitiveness

We now use Lemma 1 to derive consequences for best-possible competitive ratios of collaborative tree exploration algorithms. In the proofs below, log is always to the natural base $e$.

**Theorem 1.** *Let $c$ be any positive integer constant. Then for every $n$ and every $1 \leq k \leq n \log^c n$ there is some $D = D(n, k, c)$ such that the following holds: for any given deterministic exploration strategy with $k$ agents, there exists a tree $T$ on $n$ vertices and with height $D$ on which the strategy needs*

$$\Omega\Big( \frac{\log k}{\log \log k} \cdot (n/k + D) \Big)$$

*rounds.*

*Proof.* By the result of Dynia et al. [16] it suffices to consider the case where $k \geq \sqrt{n}$. Let $c > 0$ be a constant and assume $k \leq n \log^c n$. We apply Lemma 1 with $m = \lceil \frac{\log n}{(8+c)\log\log n} \rceil$ and $L = \lceil n/(mk) \rceil$. Using $k \geq \sqrt{n}$, we have $L = \mathcal{O}(\sqrt{n})$ and $m = o(\log n)$ and thus $n \geq L \cdot 16^m$ holds for sufficiently large $n$. The lemma states that if

$$k \leq \frac{n^{1+1/m}}{6L(m+1)^2(2L)^{1/m}} \tag{5}$$

then there is a tree of height $D := Lm$ on which the strategy needs at least $L\binom{m}{2} = \Omega((n/k + D) \cdot \log k / \log\log k)$ rounds. To complete the proof, we need to show that (5) holds for all $1 \leq k \leq n \log^c n$. We split the analysis to two cases. Let us first assume $k \geq n/m$ and thus $L = 1$. This implies

$$\frac{n^{1+1/m}}{6L(m+1)^2(2L)^{1/m}} \geq \frac{n^{1+1/m}}{24m^2} \geq \frac{n \log^{8+c} n}{24 \log^2 n} \geq k,$$

when $k \leq n \log^c n$ and for sufficiently large $n$.

Now we consider the case $k < n/m$. Using that assumption and the definition of $L$ we obtain $L(m+1)^2 \leq 4mn/k$ and $2L \leq 4n/(mk)$. Putting it all together we have

$$\frac{n^{1+1/m}}{6L(m+1)^2(2L)^{1/m}} = \frac{n}{6L(m+1)^2} \left(\frac{n}{2L}\right)^{1/m} \geq \frac{k}{24m} \left(\frac{mk}{4}\right)^{1/m} \geq k,$$

where the last inequality holds for $k \geq \sqrt{n}$ because, for sufficiently large $n$,

$$(mk)^{1/m} \geq k^{1/m} \geq e^{\frac{\log n}{2m}} \geq e^{\frac{(8+c)\log\log n}{4}} \geq (\log n)^2 \geq 100m.$$

□

**Theorem 2.** *Given any constant $\varepsilon > 0$ there is an integer $D = D(\varepsilon)$ such that for sufficiently large $n$ and for every deterministic exploration strategy using $k \leq D \cdot n^{1+\varepsilon}$ agents, there exists a tree on $n$ vertices and with height $D$ on which the strategy needs at least $D/(5\varepsilon)$ rounds.*

*Proof.* We choose $L = 1$ and $m = \lceil 1/2\varepsilon \rceil$ in Lemma 1. The claim is trivial unless $\varepsilon < 1/5$, so we can eliminate rounding and assume generously that $1/m \geq 1.4\varepsilon$. The condition $n \geq L \cdot 16^m$ is clearly satisfied for sufficiently large $n$.

By Lemma 1, there is a tree $T$ of height $m$ that needs time $\binom{m}{2} \geq m/(5\varepsilon)$ to be explored, provided the team has size at most (for $n$ sufficiently large)

$$k \leq n^{1+1.4\varepsilon}/(12(m+1)^2) \leq m \cdot n^{1+\varepsilon} = D \cdot n^{1+\varepsilon}.$$

□

**Theorem 3.** *For every $n$ and every deterministic exploration strategy using $k = n$ agents, there exists a tree $T$ on $n$ vertices and with height $D = \omega(1)$ such that the strategy needs at least $D^2/3$ rounds to explore $T$.*

*Proof.* We choose $L = 1$ and $m = \lceil \sqrt{\log n} \rceil$ in Lemma 1. Then $n \geq L \cdot 16^m$ holds for sufficiently large $n$. Note also that for sufficiently large $n$,

$$\frac{n^{1+1/m}}{12(m+1)^2} = \Omega(n \cdot e^{\sqrt{\log n}} / \log n) \geq n.$$

The lemma now states that there exists a tree $T$ of height $m$ such that the given strategy with $k = n$ agents needs at least $\binom{m}{2}$ rounds to explore $T$. Since for large enough $n$ we have $\binom{m}{2} \geq m^2/3$, this implies the theorem. □

**Theorem 4.** *For any function $D \leq n^{o(1)}$ and any exploration strategy using $k = \Theta(n)$ agents, the competitive ratio on the set of all trees of size $n$ and height at least $D$ is $\omega(1)$.*

*Proof.* Suppose that $D \leq n^{o(1)}$, i.e., $D = n^{1/f(n)}$, where $f(n)$ is a function which tends to infinity with $n$. Let $L = D$ and note that we have

$$\frac{16L^{1/m}}{n^{1/m}} \leq \frac{L^{1+1/m}(m+1)^2}{n^{1/m}} \leq \frac{4m^2 n^{2/f(n)}}{n^{1/m}}.$$

If we choose $m = m(n) = \omega(1)$ as a function growing sufficiently slowly such that we have $m \leq \min\{(f(n))^{1/2}, (\log n)^{1/2}\}$, then the following is true:

$$\frac{4m^2 n^{2/f(n)}}{n^{1/m}} = 4 \cdot e^{2\log m + 2(\log n)/f(n) - \log n/m} \to 0.$$

This implies $16L^{1/m} = o(n^{1/m})$ and $L^{1+1/m}(m+1)^2 = o(n^{1/m})$. In particular, $n \geq L \cdot 16^m$ for sufficiently large $n$. Moreover, if $n$ is large enough then $k = \Theta(n)$ implies

$$\frac{n^{1+1/m}}{6L(m+1)^2(2L)^{1/m}} = \frac{n^{1+1/m}}{o(n^{1/m})} \geq k.$$

By Lemma 1, there exists a tree $T$ with height $Lm \geq D$ on which the strategy needs $L\binom{m}{2} = \omega(Lm)$ rounds. Since $k = \Theta(n)$, the offline optimum is $\mathcal{O}(Lm + n/k) = \mathcal{O}(Lm)$, so the competitive ratio on the set of trees of height at least $D$ is $\omega(1)$, as claimed. □

## 6 Conclusions

In this paper we presented new lower bounds for collaborative tree exploration. Including our results, the following bounds are now known. For $k = \mathcal{O}(1)$ or $k \geq D \cdot n^{1+\varepsilon}$ agents, a competitive ratio of $\Theta(1)$ can be achieved. For $\omega(1) \leq k \leq n \log^c n$, the best-possible competitive ratio is bounded by $\Omega(\log k / \log\log k)$, and no constant competitive ratio is possible when $n \log^c n \leq k \leq D \cdot n^{1+o(1)}$. On the other hand, the best exploration algorithms for trees in the domain $k \leq D \cdot n^{1+o(1)}$ stay close to the trivial competitive ratio of $k$ (the best ratios are $k/\log k$ and $k^{o(1)}$, depending on the domain).

In summary, we now fully understand the domain where constant competitive ratios are possible, but, outside this domain, a wide gap persists.

**Acknowledgments.** We would like to thank Rajko Nenadov for useful discussions.

## References


1. Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM J. Comput. **29**(4), 1164–1188 (2000)
2. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS), pp. 218–223 (1979)
3. Alon, N., Avin, C., Koucký, M., Kozma, G., Lotker, Z., Tuttle, M.R.: Many random walks are faster than one. Comb. Probab. Comput. **20**(4), 481–502 (2011)
4. Ambühl, C., Gąsieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. ACM Trans. Algorithms **7**(2), 1–21 (2011)
5. Awerbuch, B., Betke, M., Rivest, R.L., Singh, M.: Piecemeal graph exploration by a mobile robot. Inf. Comput. **152**(2), 155–172 (1999)
6. Bender, M.A., Fernández, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: exploring and mapping directed graphs. Inf. Comput. **176**(1), 1–21 (2002)
7. Bender, M.A., Slonim, D.K.: The power of team exploration: two robots can learn unlabeled directed graphs. In: Proceedings of 35th Annual Symposium on Foundations of Computer Science (FOCS), pp. 75–85 (1994)
8. Blum, M., Kozen, D.: On the power of the compass (or, why mazes are easier to search than graphs). In: Proceedings of the 19th Annual Symposium on Foundations of Computer Science (FOCS), pp. 132–142 (1978)
9. Chalopin, J., Das, S., Disser, Y., Mihalák, M., Widmayer, P.: Mapping simple polygons: how robots benefit from looking back. Algorithmica **65**(1), 43–59 (2011)
10. Chalopin, J., Das, S., Disser, Y., Mihalák, M., Widmayer, P.: Mapping simple polygons. ACM Trans. Algorithms **11**(4), 1–16 (2015)
11. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. J. Graph Theory **32**(3), 265–297 (1999)
12. Dereniowski, D., Disser, Y., Kosowski, A., Pająk, D., Uznański, P.: Fast collaborative graph exploration. Inf. Comput. **243**, 37–49 (2015)
13. Diks, K., Fraigniaud, P., Kranakis, E., Pelc, A.: Tree exploration with little memory. J. Algorithms **51**(1), 38–63 (2004)
14. Disser, Y., Hackfeld, J., Klimm, M.: Undirected graph exploration with $\Theta(\log \log n)$ pebbles. In: Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 25–39 (2016)
15. Duncan, C.A., Kobourov, S.G., Kumar, V.S.A.: Optimal constrained graph exploration. ACM Trans. Algorithms **2**, 380–402 (2006)
16. Dynia, M., Łopuszański, J., Schindelhauer, C.: Why robots need maps. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 41–50. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72951-8_5
17. Elsässer, R., Sauerwald, T.: Tight bounds for the cover time of multiple random walks. Theor. Comput. Sci. **412**(24), 2623–2641 (2011)
18. Fraigniaud, P., Gąsieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks **48**(3), 166–177 (2006)
19. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. **345**(2–3), 331–344 (2005)
20. Higashikawa, Y., Katoh, N., Langerman, S., Tanigawa, S.I.: Online graph exploration algorithms for cycles and trees by multiple searchers. J. Comb. Optim. **28**(2), 480–495 (2012)

21. Hoffmann, F.: One pebble does not suffice to search plane labyrinths. In: Proceedings of the 3rd International Symposium on Fundamentals of Computation Theory (FCT), pp. 433–444 (1981)
22. Ortolf, C., Schindelhauer, C.: Online multi-robot exploration of grid graphs with rectangular obstacles. In: Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 27–36 (2012)
23. Ortolf, C., Schindelhauer, C.: A recursive approach to multi-robot exploration of trees. In: Halldórsson, M.M. (ed.) SIROCCO 2014. LNCS, vol. 8576, pp. 343–354. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09620-9_26
24. Ortolf, C., Schindelhauer, C.: Strategies for parallel unaware cleaners. Theor. Comput. Sci. **608**, 178–189 (2015)
25. Reingold, O.: Undirected connectivity in log-space. J. ACM **55**(4), 1–24 (2008)

# Wireless Evacuation on $m$ Rays with $k$ Searchers

Sebastian Brandt[1], Klaus-Tycho Foerster[2(✉)], Benjamin Richner[1], and Roger Wattenhofer[1]

[1] ETH Zürich, Zürich, Switzerland
{brandts,wattenhofer}@ethz.ch, benri@bluewin.ch
[2] Aalborg University, Aalborg, Denmark
ktfoerster@cs.aau.dk

**Abstract.** We study the online problem of evacuating $k$ robots on $m$ concurrent rays to a single unknown exit. All $k$ robots start on the same point $s$, not necessarily on the junction $j$ of the $m$ rays, move at unit speed, and can communicate wirelessly. The goal is to minimize the competitive ratio, i.e., the ratio between the time it takes to evacuate all robots to the exit and the time it would take if the location of the exit was known in advance, on a worst-case instance.

When $k = m$, we show that a simple waiting strategy yields a competitive ratio of 4 and present a lower bound of $2 + \sqrt{7/3} \approx 3.52753$ for all $k = m \geq 3$. For $k = 3$ robots on $m = 3$ rays, we give a class of parametrized algorithms with a nearly matching competitive ratio of $2 + \sqrt{3} \approx 3.73205$. We also present an algorithm for $1 < k < m$, achieving a competitive ratio of $1 + 2 \cdot \frac{m-1}{k} \cdot \left(1 + \frac{k}{m-1}\right)^{1+\frac{m-1}{k}}$, obtained by parameter optimization on a geometric search strategy. Interestingly, the robots can be initially oblivious to the value of $m > 2$.

Lastly, by using a simple but fundamental argument, we show that for $k < m$ robots, no algorithm can reach a competitive ratio better than $3 + 2\lfloor (m-1)/k \rfloor$, for every $k, m$ with $k < m$.

## 1 Introduction

Searching for an unknown target is a fundamental problem in computer science and mathematics, especially in the area of robotics. The standard toolkit to analyze this class of problems is competitive analysis [32], i.e., our goal is to design online algorithms with a small competitive ratio, which compares the performance of the online algorithm to an optimal offline solution which knows the target location beforehand.

As pointed out by Hammar et al. [22], "*A problem with paradigmatic status in this framework is searching on m concurrent rays*," which is the focus of this paper. More precisely, we study the problem of evacuating $k \leq m$ robots on $m$ concurrent rays (i.e., semi-infinite lines) to an unknown exit $z$ [23,26], with the robots communicating wirelessly [14,18].

The seminal forefather of this problem is the linear search problem, also known as the cow path problem, first posed by Beck [6] and Bellman [8]: A


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 140–157, 2017.
https://doi.org/10.1007/978-3-319-72050-0_9

searcher has to find an object of unknown location on the infinite line (i.e., 2 concurrent rays). The optimal online algorithm achieves a competitive ratio of 9, in each iteration doubling the search depth $1, 2, 4, \ldots$ on each side of the starting point $s$ [7]. Gal [21] and Baeza-Yates et al. [4] then extended their results to the model of $m$ concurrent rays, where the optimal strategy is to, instead of doubling the search depth, use a factor of $m/(m-1)$, yielding an optimal competitive ratio of $1 + 2m^m/(m-1)^{m-1}$ [29]. If $k$ robots can search for the exit, and one robot finding it terminates the search, a competitive ratio of $1 + 2(m/k - 1)/(m/(m-k))^{m/k}$ is optimal [30].

The concepts of collaborative evacuation and wireless communication are more recent additions in this field. In the case of the (unit speed) robots only being able to communicate when they meet, for $k = m$ a competitive ratio of 9 is again optimal [23] if there is a minimum distance to the exit, else $1 + 2(p+1)^{p+1}/p^p$ for $p = \lceil \log m \rceil$ is optimal. In the special case of $m = 2$ and $k > m$, 9 is optimal as well [12]. Baeza-Yates and Schott studied wireless communication in this context: Even though most of their paper "Parallel searching in the plane" [5] is about searching the plane, they also considered the evacuation problem with two searchers on the line, pointing out that a competitive ratio of 3 is then optimal for $k \geq m = 2$. Further collaborative robot evacuation studies in geometric settings have been performed by Czyzowicz et al.: Evacuating the circle with $k = 2$ [13], the line with faulty robots [17], the disk [14–16] (see also [11]), and equilateral triangles and squares [18], with [15,18] also studying wireless communication.

*Contributions.* In this paper, we extend the model of Baeza-Yates and Schott [5] beyond the infinite line (i.e., $m = 2$), by examining the problem of evacuating $1 < k \leq m$ robots on $m$ rays with wireless communication, which has not been studied before to the best of our knowledge. We also study the case that the $k$ robots do not start on the junction $j$ of the $m$ rays.

When starting on the junction with $k = m > 2$ robots, we show that a competitive ratio of 3 is still optimal, and starting away from the junction allows for a 4-competitive algorithm. For the special case of $k = m = 3$, we present a class of parametrized algorithms with a competitive ratio of $2 + \sqrt{3} \approx 3.73205$. We also give lower bounds of $2 + \sqrt{7/3} \approx 3.52753$, for every $k = m \geq 3$.

Furthermore, we consider the case of less robots than rays, i.e., collaborative wireless evacuation with $1 < k < m$ robots. Even though the $k$ robots are oblivious to the number of $m > 2$ rays, our optimization of parametrized geometric search strategy yields a competitive ratio of at most $1 + 2 \cdot \frac{m-1}{k} \cdot \left(1 + \frac{k}{m-1}\right)^{1+\frac{m-1}{k}}$. Moreover, as we show, even when starting on the junction, no algorithm can have a better competitive ratio than $3+2\lfloor (m-1)/k \rfloor$, for any $k, m$ with $k < m$.

*Paper Organization.* In the following paragraph we discuss further related work, before introducing the necessary formal preliminaries in Sect. 2. We then consider the case of $m$ robots on $m$ rays in Sect. 3, with an in-depth focus of 3 robots on 3 rays. Afterwards, we study the more general case of $1 < k < m$ robots on $m$ rays

in Sect. 4, also detailing a lower bound for $k < m$ with a simple but fundamental argument. Lastly, we conclude in Sect. 5.

*Further Related Work.* Results for the search problem on $m$ rays can be used for showing competitive bounds for search problems in various classes of simple polygons, cf. [23,29], with further applications in hybrid [26] and interruptible [1] algorithms. The classic linear search or cow path problem has moreover been studied in a multitude of models, e.g., adding turn costs [9,19] (also with multiple searchers on rays [2]), with a single [25] or multiple error prone robots [17], or a moving target [9]. Bose et al. [10] gave tight bounds on the competitive ratio with distance bounds to the target, showing that the optimal search strategy is then unique.

Searching on $m$ rays has furthermore been considered with multiple targets [3], with only one robot being allowed to move at a time [26], regarding advice complexity [24], and randomized algorithms [27,31] – cf. the survey by Tate [33] for an overview of the latter.

On graphs, the problem of finding a specified node in an online fashion is also known as treasure hunt or as the node searching problem [20,28].

## 2 Preliminaries

We consider the problem of collaboratively evacuating $k$ robots $R_0, \ldots, R_{k-1}$ on $m$ concurrent rays $a_0, \ldots, a_{m-1}$, joined at a common junction $j$. All robots start at the same point $s$, w.l.o.g. on ray $a_0$, where $s$ does not have to be the junction $j$. All robots have to reach the single exit $z$ on some ray $a_z$, the location and ray of $z$ is unknown until one robot reaches the location of the exit $z$. We denote the distance of the junction $j$ to the start $s$ by $\overline{js}$. The robots have the same unit maximum speed and can communicate wirelessly, instantaneously sharing their information. As thus, we can assume that one central algorithm controls all robots. Unless otherwise noted, we assume that the robots travel at unit speed when moving.

The goal is to minimize the time needed for all robots to reach the exit, compared to the minimum time needed if all information about the environment would be revealed initially. Hence, we study this problem using competitive analysis: The competitive ratio of an online (evacuation) algorithm is measured as the supremum of the ratio of the time needed for all robots to reach the exit and the distance $Z$ from $s$ to $z$, for all start and exit locations.

If the distance between the start $s$ and the exit $z$ is allowed to be arbitrarily small, no online algorithm (without infinitesimal steps) can achieve a constant competitive ratio for $k < m$: As thus, we use the common assumption of at least unit distance between start $s$ and exit $z$, cf. [1].

## 3 $m$ Robots on $m$ Rays

We start our study of robot evacuation by considering one robot for each ray. In Subsect. 3.1 we gather some basic observations. Note that Observations 1 and 2

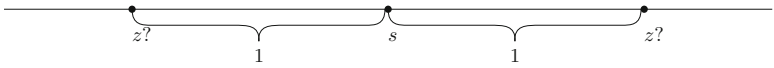



**Fig. 1.** As the robots starting on $s$ are oblivious to the direction of the exit $z$, both points of distance 1 need to be explored by at least one robot, meaning that at least one robot takes a time of $t = 3$ to reach the exit (in this case all robots can also evacuate the graph at a time of $t = 3$).

can be found with similar arguments for $k = 2$ in [5]. We further examine the case of 3 robots on 3 rays in Subsect. 3.2.

### 3.1 The General Case of $m$ Robots on $m$ Rays

If all $m$ robots start *on* the junction $j$, then each robot $R_i$ can explore ray $a_i$ at unit speed, with some robot finding $z$ at time $t = Z$. Then, all other robots are at distance $2Z$ from $z$, inducing a total evacuation time of $t = 3Z$ if they all directly travel to the exit. Trivially, in the case of $k = 1$, a single robot starting on the end of a single ray will find the exit in optimal time.

**Observation 1.** *Let $s = j$ and $k = m$, with $m > 1$. There exists an online algorithm evacuating the $m$ robots with a competitive ratio of 3.*

For $m > 1$, no better ratio than 3 is possible (cf. also Fig. 1): Assume all $2 \leq k \leq m$ robots start on the junction $j$ and the exit is at distance $Z = 1$. In the worst case, the exit will be on the last ray explored until distance 1 (which could coincide with the first ray being explored until distance 1), so at least one robot will need a time of $t = 3Z$ to reach the exit $z$.

**Observation 2.** *For every $2 \leq k \leq m$: No online algorithm can achieve a better competitive ratio than 3 for evacuating the $k$ robots.*

The situation is more difficult when the robots do not start on the junction $j$ and $m > 2$.[1] If we knew the initial direction of the junction, we could send $m - 1$ robots there, again obtaining a competitive ratio of 3 as before.

The following algorithm yields an upper bound of 4 for the competitive ratio even when the direction of the junction is not known: Send two robots $R_0, R_1$ in opposing directions until either the exit $z$ or the junction $j$ is found, with the remaining $m - 2$ robots waiting at the start $s$. If the exit $z$ is found first (or simultaneously), a competitive ratio of 3 can again be achieved by directly sending all robots to the exit $z$. If the junction is found first, we stop the robots $R_0, R_1$ for a duration of $\overline{js}$, while the other $m - 2$ robots travel to the junction. We then proceed as if $s$ was the point from which all rays emanate and the section between $s$ and $j$ was actually comprised of the first parts of $m - 1$ rays that just happened to be glued together. According to this equivalent consideration, at

[1] If $m = 2$, then a competitive ratio of 3 can be reached again, as every point can be seen as the junction.

time $2\overline{js}$, all robots are on their rays at distance $\overline{js}$ from $s$ and then continue to explore their assigned rays. When the exit $z$ is found by one robot at time $\overline{js} + Z$, all other robots move to the exit $z$ in time $2Z$, obtaining a competitive ratio of $(\overline{js} + 3Z)/Z < 4$.

**Observation 3.** *Let $k = m$, $m > 2$. There exists an online algorithm evacuating the $m$ robots with a competitive ratio of at most 4.*

We will later show a lower bound of $2 + \sqrt{7/3} \approx 3.52753$ in Corollary 2, for all $k = m \geq 3$.

### 3.2 The Case of 3 Robots on 3 Rays

We start with a lower bound for the competitive ratio of evacuating 3 robots from 3 rays, before giving a nearly matching upper bound in Theorem 3.

**Theorem 1 (Lower bound of $2 + \sqrt{7/3}$ for 3 robots on 3 rays).** *No online algorithm can achieve a better competitive ratio than $2 + \sqrt{7/3} \approx 3.52753$ for evacuating 3 robots on 3 rays.*

*Proof.* As evacuating 3 robots on 3 rays has a competitive ratio of 3 when $s = j$, we assume that $s \neq j$, $s \in a_0$, and $Z > \overline{js}$. Also, we can assume in a worst-case fashion that the junction $j$ lies on the side of $s$ that ensures that at time $\overline{js}$ at most one of the three robots is closer to $j$ than in the beginning, i.e., closer to $j$ than $\overline{js}$.

It follows that the earliest time when the 2 points of distance $3/2 \cdot \overline{js}$ from $s$ on $a_1, a_2$ have been visited is at time $5/2 \cdot \overline{js}$: Only the robot that is (possibly) closer to $j$ at time $\overline{js}$ than in the beginning can visit any of these 2 points before time $5/2 \cdot \overline{js}$; however, since it can visit the first of the two at time $3/2 \cdot \overline{js}$ at the earliest, it cannot visit the other one before time $5/2 \cdot \overline{js}$.

W.l.o.g., let $R_2$ be a robot who has (possibly previously) visited a point $p$ farthest away from the junction on the starting ray $a_0$ at time $t = 5/2 \cdot \overline{js}$. We will now show Theorem 1 by case distinction for a point $y$, denoting where $R_2$ is at time $5/2 \cdot \overline{js}$. The case distinction will depend on a "border"-value $b$, later to be optimized. We refer to Fig. 2 for an overview of the construction.

We start with the first case of $\overline{jy} \geq b + \overline{js}$ and $y$ lies on $a_0$: Then, we place the exit $z$ at one of the points of distance $3/2 \cdot \overline{js}$ from $s$ on $a_1, a_2$ that is visited last by the strategy used by the three robots. As the exit cannot have been found before time $5/2 \cdot \overline{js}$, robot $R_2$ will need (in the best case) $5/2 \cdot \overline{js} + \overline{sy} + 3/2 \cdot \overline{js} = 4 \cdot \overline{js} + \overline{sy} \geq 4 \cdot \overline{js} + b$ total time to reach the exit $z$. Note that in this case, the optimal time is $Z = 3/2 \cdot \overline{js}$.

Next, we consider the second and remaining case of $\overline{jy} < b + \overline{js}$ or $y$ not lying on $a_0$. To still reach $y$ at time $5/2 \cdot \overline{js}$, $R_2$ could have moved at most to a $p$ with $\overline{ps} \leq 5/4 \cdot \overline{js} + b/2$. We now place the exit $z$ a distance of $\varepsilon$ "behind" one of the three points of distance $5/4 \cdot \overline{js} + b/2$ to the start $s$ which will be reached last. Note that, as shown before, the earliest time when both of these points on $a_1, a_2$ can be reached is at time $5/2 \cdot \overline{js} + b/2 + \varepsilon$, and the earliest time when

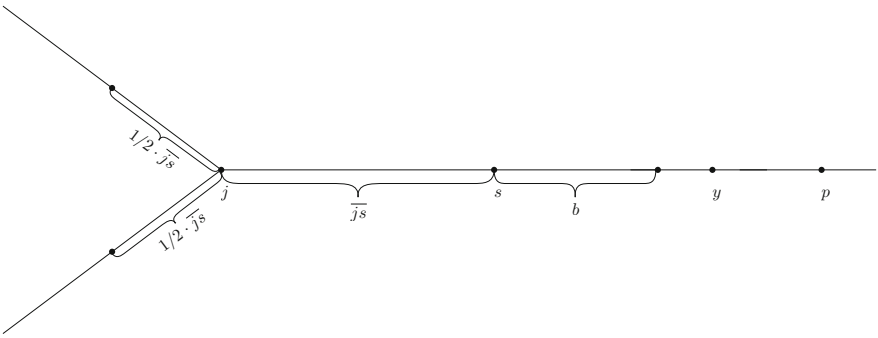



**Fig. 2.** The robots $R_1, R_2, R_3$ start on $s$ and have to find the unknown exit $z$. Point $p$ depicts the farthest any robot has been away from the junction on the starting ray $a_0$ until time $5/2 \cdot \overline{js}$, and $y$ where a robot visiting $p$ is at time $5/2 \cdot \overline{js}$. Depending on if $y$ is at least $\overline{js}+b$ away from the junction $j$ or not, we give two different arguments in the proof of Theorem 1, resulting in a (normalized to $\overline{js}$) value of $b$ of $-1+\sqrt{21}/2 \approx 1.29129$ and a lower bound of $2+\sqrt{7/3} \approx 3.52753$.

the respective point on $a_0$ can be reached (assuming both points on $a_1, a_2$ were reached) is not before time $5/2 \cdot \overline{js} + 5/4 \cdot \overline{js} + b/2 + \varepsilon - b$. As thus, for all robots to evacuate to the exit, a time of at least $5/2 \cdot \overline{js} + (5/4 \cdot \overline{js} + b/2 + \varepsilon - 3/2) + 5/4 \cdot \overline{js} + b/2 + \varepsilon + 5/4 \cdot \overline{js} + b/2 + \varepsilon = 19/4 \cdot \overline{js} + 3/2 \cdot b + 3 \cdot \varepsilon$ is needed, with the optimal solution taking time $Z = 5/4 \cdot \overline{js} + b/2 + \varepsilon$.

To optimize the lower bound in respect to $b$, we solve $\frac{19/4 \cdot \overline{js} + 3/2 \cdot b + 3 \cdot \varepsilon}{5/4 \cdot \overline{js} + b/2 + \varepsilon} = \frac{4 \cdot \overline{js} + b}{3/2 \cdot \overline{js}}$ for $b$. By normalizing $\overline{js}$ to unit value and restricting $b > 0$, solving the above equation gives us the parameter $b = -1 - \epsilon + 1/2 \cdot \sqrt{21 + 12\varepsilon + 4\varepsilon^2}$, which is approximately 1.29129 for $\varepsilon \to 0$ for our proof, as the functions defined by the terms on the individual sides of the equation are monotonically decreasing and increasing, respectively.

Observe that $-1 - \epsilon + 1/2 \cdot \sqrt{21 + 12\varepsilon + 4\varepsilon^2}$ is monotonically decreasing when considered as a function of $\varepsilon$, i.e., for all values of $\varepsilon > 0$, we obtain the supremum at $-1 + \sqrt{21}/2 \approx 1.29129$.

Therefore, we achieve a lower bound of $\frac{4 - 1 + \sqrt{21}/2}{3/2} = 2 + \sqrt{7/3} \approx 3.52753$. □

We note that the construction from the above proof can be extended to $k = m > 3$ robots and rays, as at time $t = \overline{js}$, at most $\lfloor m/2 \rfloor$ robots can be guaranteed to be at the junction $j$.

**Corollary 2.** *For every $k = m \geq 3$ holds: No online algorithm can achieve a better competitive ratio than $2 + \sqrt{7/3} \approx 3.52753$ for evacuating $k = m$ robots on $m$ rays.*

We now give an algorithm with a nearly matching competitive ratio for 3 robots:

**Theorem 3.** *There exists an online algorithm evacuating 3 robots on 3 rays with a competitive ratio of $2 + \sqrt{3} \approx 3.73205$.*

*Proof.* We know from Observation 1 that there is an algorithm with a competitive ratio of 3 when starting on the junction $j$, so suppose that $j \neq s$. We prove Theorem 3 by giving a whole class of algorithms, all reaching the desired competitive ratio. To describe these strategies, we develop a parametrized approach by composing an algorithm that moves the robots according to certain parameters and then optimizing the competitive ratio over the parameter space. More specifically, the algorithm depends on two parameters $\alpha$ and $\beta$ which are constrained by the inequalities $0 \leq \beta \leq \alpha \leq \frac{1}{2}$ and $2\alpha \leq \beta + \frac{1}{2}$ and moves the three robots $R_0, R_1, R_2$ as described in the following. We note that if one robot finds the exit, all the other robots abandon their strategy and take the shortest path to the exit $z$.

Figure 3 serves as a visual aid to understand the parameters and the respective strategies. We send $R_0$ in one direction, $R_1$ in the other, and $R_2$ waits until the junction $j$ (or the exit) is found. W.l.o.g. suppose $R_0$ reaches the junction $j$ after $\overline{js}$ time passed, i.e., $R_1$ is at distance $\overline{js}$ to $s$ on the other side of ray $a_0$, and $R_2$ is still on the start $s$. Then, $R_0$ moves for $\alpha \cdot \overline{js}$ time into one of the two branching rays $a_1, a_2$, returns back to the junction $j$, and moves into the other ray of $a_1, a_2$. Meanwhile, at time $\overline{js}$, $R_1$ starts to move deeper into the ray $a_0$ away from the junction $j$ by $\beta \cdot \overline{js}$ before turning around and walking backwards until it reaches the same distance to the junction $j$ as $r_3$, which starts moving towards the junction at time $\overline{js}$ and then moves into the ray $R_0$ explored first (and left by the time $R_2$ arrives at the junction). The three robots continue to move straight to a distance of $\overline{js} + \beta \cdot \overline{js}$ to $s$ on their respective ray, and those that arrive early wait for the others. Then, they all move uniformly outwards at equal distance to the start $s$.

We will now start analyzing the competitive ratio of the above algorithm: Until the junction is found, any exit found will lead to a competitive ratio of 3. Observe that until all three robots move outwards from the start $s$ on the three rays, the following three points, with additional $\varepsilon$ distance to $s$, are worst case points regarding the competitive ratio of the algorithm (cf. Fig. 3), i.e., the time when a robot visits them for the first time will determine the competitive ratio: $p_2$, the point where $R_0$ turns around to go back to the junction, $p_1$, the point where $R_1$ turns around to go back to the start, and $p_0$, the junction itself. After that, the competitive ratio of any exit placement can only be lower, as now any remaining distance of $x$ to the exit will be covered in $x$ time by one robot.

For ease of readability, we are going to omit the additional $\varepsilon$s in the following calculations, as we are later going to consider the supremum of the competitive ratio anyway.

The three points will be reached at the following times: $p_0$ at time $\overline{js} + 2 \cdot \alpha\overline{js}$ by $R_0$, $p_1$ at time $2 \cdot \overline{js} + \beta\overline{js}$ by $R_1$, and $p_2$ at time $2 \cdot \overline{js} + \alpha\overline{js}$. If all other robots divert directly to the exit $z$ when it is found, they will reach the exit with the following additional time: $p_0$ with time $2 \cdot \overline{js} - 2 \cdot \alpha\overline{js} + 2 \cdot \beta\overline{js}$, $p_1$ with time $2 \cdot \overline{js} + 2 \cdot \beta\overline{js}$, and $p_2$ with time $2 \cdot \overline{js} + 2 \cdot \alpha\overline{js}$.

Hence, the competitive ratio induced by the three points is adding both times above, divided by the distance of the exit to the start, i.e., $3 + 2 \cdot \beta$ for $p_0$, $3 + \frac{1}{1+\beta}$ for $p_1$, and $3 + \frac{1}{1+\alpha}$ for $p_2$. Note that $3 + \frac{1}{1+\alpha} \leq 3 + \frac{1}{1+\beta}$ due to initially choosing $\beta \leq \alpha$.

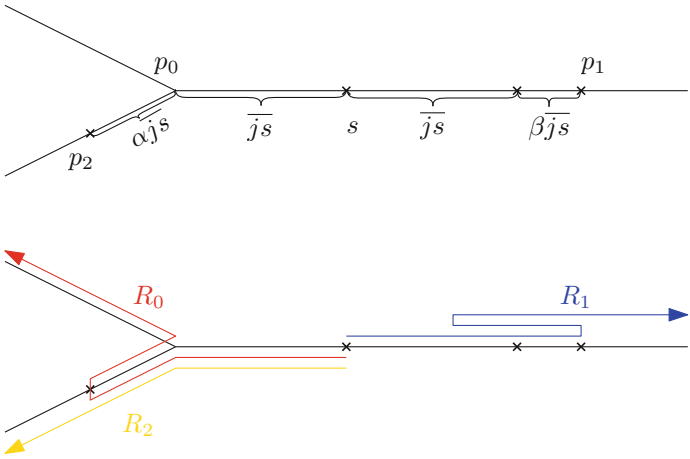



**Fig. 3.** A depiction of the parameters $\alpha$ and $\beta$ on the 3-ray and the strategies of the 3 robots (waiting is not indicated). The three worst case points $p_0, p_1, p_2$ are also marked.

As $3+2\cdot\beta$ is strictly monotonically increasing and $3+\frac{1}{1+\beta}$ strictly monotonically decreasing, the desired solution can be obtained by equalizing both terms in the parameter range, with $\beta = \frac{\sqrt{3}-1}{2}$. As $\alpha$ can be chosen freely in the parameter space, we have generated a whole class of algorithms with identical competitive ratio of $\min_\beta \max\left(3+2\cdot\beta, 3+\frac{1}{1+\beta}\right) = 2+\sqrt{3}$. □

## 4 $1 < K < m$ Robots on $m$ Rays

In this section, we continue our study of collaborative robot evacuation by considering the case of $1 < k < m$ robots on $m$ rays. Since in this case the number of robots is not sufficient anymore to assign a ray to each robot, a more intricate scheme than before is required in order to achieve a good competitive ratio. In the literature, similar problems have been tackled by using geometric search. We show that this general idea can also be applied in our setting and present an upper bound for the competitive ratio where the factor that governs the exponential growth is chosen in a way that minimizes the bound. We complement this result with a lower bound for all wireless evacuation algorithms where $k < m$.

### 4.1 An Upper Bound on the Competitive Ratio

We start by developing GeomSearch$(\alpha, \beta)$, an algorithm for evacuating $k$ robots from $m$ rays where the robots start in the junction $j$. The algorithm depends on two parameters $\alpha$ and $\beta$ which we will determine later. It proceeds as follows:

Each robot explores the $m$ rays in so-called *exploration steps* where each exploration step consists of exploring some ray up to some depth and then returning to the junction. More specifically, robot $R_i$ starts by exploring ray

$a_i$ up to depth $\alpha\beta^i$ upon which it returns to the junction. Then it explores ray $a_{i+k \pmod m}$ up to depth $\alpha\beta^{i+k}$, returns to the junction, explores ray $a_{i+2k \pmod m}$ up to depth $\alpha\beta^{i+2k}$, and so on. In other words, robot $R_i$ performs its $q$th exploration step on ray $a_{i+(q-1)k \pmod m}$ with a depth of $\alpha\beta^{i+(q-1)k}$. Note that in each exploration step of robot $R_i$ the explored depth increases by a factor of $\beta^k$ and that it always chooses the ray to be explored next by increasing the ray index by $k$ (modulo $m$). If a robot finds the exit $z$ it immediately informs all other robots, upon which each robot immediately aborts its exploration and heads straight for $z$.

From the definition of the exploration steps it follows that for any two robots $R_h$ and $R_i$ with $h < i$ and any positive integer $q$, the $q$th exploration step of $R_i$ takes strictly more time than the $q$th exploration step of $R_h$ and the $(q+1)$th exploration step of $R_h$ takes strictly more time than the $q$th exploration step of $R_i$. Thus, we obtain the following observation which sheds light on the order in which the robots take their exploration steps.

**Observation 4.** *Let $h$, $i$ and $q$ be integers satisfying $0 \le h < i \le k-1$ and $q \ge 1$. Then robot $R_h$ finishes its $q$th exploration step before $R_i$ finishes its $q$th exploration step, and $R_i$ finishes its $q$th exploration step before $R_h$ finishes its $(q+1)$th exploration step.*

In order to prove an upper bound on the competitive ratio of Algorithm GeomSearch$(\alpha, \beta)$ for suitably chosen $\alpha$ and $\beta$, we need a technical lemma, given in the following.

**Lemma 4.** *Let* $\beta = \left(1 + \frac{k}{m-1}\right)^{1/k}$. *Then* $1 + 2\dfrac{\beta^{m+k-1}}{\beta^k - 1} \ge 3 + 2\dfrac{\beta^m}{\beta^k - 1}$ .

*Proof.* For a contradiction, assume that the statement is false. We obtain the following series of implications:

$$
\begin{aligned}
& 3 + 2\frac{\beta^m}{\beta^k - 1} > 1 + 2\frac{\beta^{m+k-1}}{\beta^k - 1} \\
\Longrightarrow \quad & \beta^k - 1 > \beta^m(\beta^{k-1} - 1) \\
\Longrightarrow \quad & \frac{k}{m-1} > \left(1 + \frac{k}{m-1}\right)^{m/k} \left(\left(1 + \frac{k}{m-1}\right)^{(k-1)/k} - 1\right) \\
\Longrightarrow \quad & \frac{k}{m-1} > \left(1 + \frac{m}{m-1}\right) \left(\left(1 + \frac{k}{m-1}\right)^{(k-1)/k} - 1\right) \\
\Longrightarrow \quad & \frac{2m+k-1}{2m-1} > \left(\frac{m+k-1}{m-1}\right)^{(k-1)/k} \\
\Longrightarrow \quad & \frac{2m-1}{2m+k-1} < \left(\frac{m-1}{m+k-1}\right)^{(k-1)/k} = \left(1 + \frac{-k}{m+k-1}\right)^{(k-1)/k}
\end{aligned}
$$

$$\implies \quad \frac{2m-1}{2m+k-1} < 1 + \frac{-(k-1)}{m+k-1} = \frac{m}{m+k-1}$$
$$\implies \quad mk + 1 < 2m + k$$

For the third and sixth implication we used the generalized version of Bernoulli's inequality which says that for any two real numbers $b > -1$ and $c \geq 0$ it holds that $(1+b)^c \geq 1 + bc$ if $c \geq 1$, and $(1+b)^c \leq 1 + bc$ if $0 \leq c \leq 1$. Since $m > k \geq 2$, the obtained statement implies $k = 2$. Going back to the result after the fourth implication and plugging in $k = 2$, we obtain the following new implications:

$$\frac{2m+1}{2m-1} > \left(\frac{m+1}{m-1}\right)^{1/2} \implies (2m+1)^2(m-1) > (2m-1)^2(m+1) \implies -1 > 1.$$

We obtain a contradication, which proves the lemma statement. □

Now we can finally prove the desired upper bound.

**Theorem 5.** *Let* $\beta = \left(1 + \frac{k}{m-1}\right)^{1/k}$ *and let* $\alpha$ *be chosen such that* $\alpha\beta^{m-1} < 1$. *Then the competitive ratio of Algorithm GeomSearch*$(\alpha, \beta)$ *is at most*

$$1 + 2 \cdot \frac{m-1}{k} \cdot \left(1 + \frac{k}{m-1}\right)^{1+\frac{m-1}{k}}.$$

*Proof.* Let $R_h$ be the robot that finds the exit and assume that $R_h$ finds the exit in its $q$th exploration step. It follows from the design of our algorithm that the exit lies on ray $a_{h+(q-1)k \pmod m}$. Note that since $\alpha\beta^{m-1} < 1$ and the exit has a distance of at least 1 from the junction, we have that $q \geq 2$. Let $t_0$ denote the point in time at which $R_h$ reaches the point on $a_{h+(q-1)k \pmod m}$ with largest depth that has been explored before by some robot. Let $\Delta t$ denote the time $R_h$ travels on $a_{h+(q-1)k \pmod m}$ between $t_0$ and finding the exit, i.e., $R_h$ finds the exit at time $t_0 + \Delta t$. Furthermore, for each $R_i$ with $i \neq h$, let $E_i$ denote the exploration step $R_i$ is performing at the time when $R_h$ starts its $q$th exploration step, and let $t_1(i)$ denote the point in time at which $R_i$ finishes exploration step $E_i$. We note that at time $t_0$, the distance between $R_h$ and the junction is the depth of the previous exploration step of a robot on ray $a_{h+(q-1)k \pmod m}$ which is $\alpha\beta^{h+(q-1)k-m}$, by the definition of the exploration steps.[2] Now, we consider two cases for each robot $R_i$:

First, consider the case that $t_0 \geq t_1(i)$. Then, at time $t_0$, $R_i$ has finished exploration step $E_i$ and has started with its next exploration step. This implies that the distance between $R_i$ and the junction at time $t_0$ is smaller than the distance between $R_h$ and the junction at time $t_0$, i.e., smaller than $\alpha\beta^{h+(q-1)k-m}$. Thus, at time $t_0 + \Delta t$, the distance between $R_i$ and the junction is smaller than $\alpha\beta^{h+(q-1)k-m} + \Delta t$. We conclude that it takes $R_i$ at most $2(\alpha\beta^{h+(q-1)k-m} + \Delta t)$

[2] Here we implicitly use that $\alpha\beta^{m-1} < 1$ which ensures that the ray on which $R_h$ finds the exit, has been previously explored by some robot.

time to reach the exit after the exit has been found at time $t_0 + \Delta t$. Since $R_h$ finishes its first $q-1$ exploration steps in time

$$\sum_{x=0}^{x=q-2} 2\alpha\beta^{h+xk} = 2\alpha\beta^h \sum_{x=0}^{x=q-2} \left(\beta^k\right)^x = 2\alpha\beta^h \frac{\beta^{(q-1)k}-1}{\beta^k-1}$$

and it takes $R_h$ another $\alpha\beta^{h+(q-1)k-m} + \Delta t$ time to find the exit, we hence obtain an upper bound of

$$2\alpha\beta^h \frac{\beta^{(q-1)k}-1}{\beta^k-1} + 3(\alpha\beta^{h+(q-1)k-m} + \Delta t)$$

for the time it takes $R_i$ to reach the exit.

In order to obtain an upper bound for the competitive ratio (for our first case), we divide by the length $Z$ of the shortest path from $s$ to $z$. Note that $\Delta t$ appears with a factor of 3 in the numerator whereas it appears with a factor of 1 in the denominator. Since the competitive ratio we obtain is larger than 3, making $\Delta t$ larger decreases the competitive ratio (towards 3). Hence, by setting $\Delta t = 0$, we obtain an upper bound of

$$\begin{aligned}\frac{2\alpha\beta^h \frac{\beta^{(q-1)k}-1}{\beta^k-1} + 3\alpha\beta^{h+(q-1)k-m}}{\alpha\beta^{h+(q-1)k-m}} &= 3 + 2\frac{\beta^{(q-1)k}-1}{(\beta^k-1)\,\beta^{(q-1)k-m}} \\ &= 3 + 2\frac{\beta^m}{\beta^k-1} - \frac{2}{(\beta^k-1)\,\beta^{(q-1)k-m}}\end{aligned}$$

for the competitive ratio, which implies an upper bound of $3 + 2\beta^m/(\beta^k-1)$. Note that the last simplification does not increase the upper bound more than necessary: The term $2/((\beta^k-1)\beta^{(q-1)k-m})$ can be made arbitrarily small by increasing $q$, i.e., by choosing the exit location accordingly.

Now consider the second case, namely, that $t_0 < t_1(i)$. Then, $R_i$ is still performing exploration step $E_i$ at time $t_0$. It follows that at the time the exit is found, $R_i$ is still performing $E_i$ or $R_i$ has distance at most $\Delta t$ from the junction. Thus, we can bound (from above) the total time it takes $R_i$ to reach the exit by the sum of (1) the time it takes $R_i$ to perform its exploration steps up to and including $E_i$, (2) two times $\Delta t$, which bounds the time between reaching the junction after $E_i$ and reaching the junction possibly again after being told the location of the exit and (3) $\alpha\beta^{h+(q-1)k-m} + \Delta t$, the time it takes $R_i$ to reach the exit from the junction. The first of the three summands in turn can be bounded by the time it takes $R_{h-1 \pmod m}$ to perform its exploration steps up to and including $E_{h-1 \pmod m}$, by the definition of $E_i$ and Observation 4.[3] Hence, we obtain an upper bound of

[3] For the following calculation of the upper bound, we assume for simplicity that if $h = 0$, then $R_{h-1 \pmod m}$ performs a 0th exploration step of length $\alpha\beta^{-1}$ before its 1st exploration step. Since this can only increase the upper bound, the given bound also holds if $h = 0$.

$$\sum_{x=0}^{x=q-1} 2\alpha\beta^{h-1+xk} + 2\Delta t + \alpha\beta^{h+(q-1)k-m} + \Delta t$$
$$= 2\alpha\beta^{h-1}\frac{\beta^{qk}-1}{\beta^k-1} + \alpha\beta^{h+(q-1)k-m} + 3\Delta t$$

for the time it takes $R_i$ to reach the exit. By an argumentation analogous to the one in the previous case, we obtain an upper bound of $1+2\beta^{m+k-1}/(\beta^k-1)$ for the competitive ratio. By Lemma 4, this upper bound is larger than the upper bound for the competitive ratio obtained in the first case. Now replacing $\beta$ by $(1+k/(m-1))^{1/k}$ yields the lemma statement. □

We note that the choice of $\beta$ in Theorem 5 is not arbitrary: The given $\beta$ precisely minimizes the obtained upper bound of $1+2(\beta^{m+k-1})/(\beta^k-1)$ as can be shown by taking the derivative.

Interestingly, for $k=1$, our upper bound coincides with the competitive ratio of $1+2m^m/(m-1)^{m-1}$ from the optimal search strategy for a single robot, given in [4,21].

We now extend GeomSearch$(\alpha,\beta)$ to the setting where the robots are not required to start in the junction. As we will prove, even if the robots do not know the number of rays when they start, they can still achieve a competitive ratio of at most

$$1+2\cdot\frac{m-1}{k}\cdot\left(1+\frac{k}{m-1}\right)^{1+\frac{m-1}{k}}. \tag{1}$$

Before describing the extension of GeomSearch$(\alpha,\beta)$, we present a lemma claiming that at a certain point in time during Algorithm GeomSearch$(\alpha,\beta)$, the distribution of the robots satisfies certain properties that will be of great use later on.

**Lemma 6.** *Let $x$ be some positive real number. Consider GeomSearch$(\alpha,\beta)$ for*

$$\beta=\left(1+\frac{k}{m-1}\right)^{1/k} \qquad \text{and} \qquad \alpha=\frac{x}{\left(1+\frac{k}{m-1}\right)^2}.$$

*Let $t_0$ denote the time at which $R_0$ is at the tip (i.e., exactly in the middle) of its third exploration step. Then, at time $t_0$, each robot has a distance of at most $x$ from the junction and no robot $R_i$ with $i\geq 1$ is on the same ray as $R_0$, except possibly in the junction.*

*Proof.* By the definition of the exploration steps,

$$t_0 = 2\alpha\beta^0 + 2\alpha\beta^k + \alpha\beta^{2k} > \frac{2x}{\left(1+\frac{k}{m-1}\right)} + x \geq 2x.$$

Moreover, at time $t_0$, robot $R_0$ is exactly in distance $x$ from the junction. By Observation 4, this implies that the distance of $R_i$ from the junction is at most

$x$, for any $1 \leq i \leq k-1$. The fact that at time $t_0$, $R_0$ is the only robot on the ray it currently occupies, follows directly from the definition of the exploration steps in conjunction with Observation 4. □

We call the distribution of the robots at time $t_0$ in Lemma 6 the *third distribution*. The general idea of our extended algorithm is that the robots simulate Algorithm GeomSearch$(\alpha, \beta)$ where they consider $s$ as the junction and the path between $s$ and $j$ as $m-1$ separate paths (that just happen to be glued together). In order to be able to compute the appropriate $\beta$ in Algorithm GeomSearch$(\alpha, \beta)$, they first have to determine the number of rays, which they do by exploring the ray they are on in both directions until they find the junction. At the point in time when the junction is found, the robots have already "wasted" some time; therefore they do not return to the junction and only then start the simulation of GeomSearch$(\alpha, \beta)$, but instead jump into a hypothetical execution of GeomSearch$(\alpha, \beta)$, i.e., they move to a configuration of points that will be reached by GeomSearch$(\alpha, \beta)$ (for some suitably chosen $\alpha$) at some point in time. From there, they simply follow GeomSearch$(\alpha, \beta)$. For a formally correct description of the extended version of GeomSearch$(\alpha, \beta)$ we need some notation:

Let $a_0$ be the ray on which $s$ is located and $a_1, \ldots, a_{m-1}$ the remaining $m-1$ rays. We denote the path obtained by deleting $\overline{sj}$ from $a_0$ by $a'_0$ and the paths obtained by appending $a_1, \ldots, a_{m-1}$ to $\overline{js}$ by $a'_1, \ldots, a'_{m-1}$, respectively. We may now consider $s$ as the junction of the $m$ rays $a'_0, \ldots, a'_{m-1}$. Therefore, provided we know $m$, any $m$-ray algorithm where the robots start in the junction can be simulated on our given input where $s$ plays the role of the junction. In particular, the achieved competitive ratio of the simulation on our input is the same as the competitive ratio of the simulated $m$-ray algorithm. In the following, we describe the extension of GeomSearch$(\alpha, \beta)$ more formally:

Robots $R_0$ and $R_1$ start by exploring the ray they are located on in opposite directions until one of the two finds the exit or the junction (while everyone else simply stays in $s$). If the exit is found before or at the same time as the junction, then all robots immediately travel to the exit, which yields a competitive ratio of 3 (which is clearly smaller than the term given in (1), for any $2 \leq k < m$). Thus, in the following, assume that the junction is found before the exit. W.l.o.g. assume that $R_1$ finds the junction (which happens at time $\overline{sj}$). From here, the robots move as quickly as possible to a configuration of points that corresponds to the third distribution[4] in the (hypothetical) execution of GeomSearch$(\alpha, \beta)$ on the $m$ rays $a'_0, \ldots, a'_{m-1}$ where

$$\beta = \left(1 + \frac{k}{m-1}\right)^{1/k} \qquad \text{and} \qquad \alpha = \frac{\overline{sj}}{\left(1 + \frac{k}{m-1}\right)^2}.$$

[4] Here, a detail has to be mentioned: By changing the mapping of the $m$ labels $a'_0, \ldots, a'_{m-1}$ to the $m$ actual rays, we can change which robot is on which ray. We assume that the labels are changed in a way that ensures that $R_0$ is actually on ray $a_0$ in the third distribution.

By Lemma 6 moving to this configuration from the situation where the junction has just been found takes at most time $\overline{sj}$, i.e. the robots reach this configuration in a total time of at most $2\overline{sj}$. By the proof of Lemma 6 the (hypothetical) execution of GeomSearch$(\alpha, \beta)$ needs at least time $2\overline{sj}$ to reach the third distribution, i.e., this configuration. Thus, the robots can just wait in the reached configuration until time $2\overline{sj}$ (if they should have reached their respective points early) and then simulate the execution of GeomSearch$(\alpha, \beta)$ mentioned above, thereby reaching any point at least as fast as the (original) execution of GeomSearch$(\alpha, \beta)$ and hence achieving a smaller or equal competitive ratio as the one in Theorem 5.

Here two remarks are in order: First, since we assume that the junction is closer to $s$ than the exit is to $s$, the exit can only be found after the robots moved to the third distribution. Hence, it is indeed enough to consider only the exits found (and therefore the competitive ratios achieved) during the simulation of GeomSearch$(\alpha, \beta)$. Second, so far, for the sake of the exposition, we ignored the detail that Theorem 5 actually requires $\alpha\beta^{m-1} < 1$. This can easily be remedied by dividing the current $\alpha$ repeatedly by $\beta^k$ until $\alpha\beta^{m-1} < 1$ holds. Note that Lemma 6 then still holds with an analogous argumentation. Essentially, the only resulting change in the above considerations is that it takes GeomSearch$(\alpha, \beta)$ even longer to get to the configuration of points with which the above simulation starts.

By our above considerations, we obtain the following theorem:

**Theorem 7.** *There is an extension of Algorithm GeomSearch$(\alpha, \beta)$ for the case where the robots are not required to start in the junction that achieves a competitive ratio of at most*

$$1 + 2 \cdot \frac{m-1}{k} \cdot \left(1 + \frac{k}{m-1}\right)^{1+\frac{m-1}{k}} .$$

### 4.2 A Lower Bound on the Competitive Ratio

In this section, we use a simple but fundamental technique to bound the competitive ratio for the general case of $k$ robots on $m > k$ rays from below.

**Theorem 8.** *There is no wireless evacuation algorithm for $k$ robots on $m > k$ rays that achieves a competitive ratio of less than $3 + 2\lfloor (m-1)/k \rfloor$.*

*Proof.* Set $x = \lfloor (m-1)/k \rfloor$. Consider any wireless evacuation algorithm $A$ for $k$ robots on $m > k$ rays. We assume that all robots start in the junction (which we may choose to be the case as we are going to prove a lower bound). Since $A$ solves the problem of wireless evacuation, there must be a point in time where all rays have been explored up to some depth that is strictly larger than 1, provided that the exit has not been found so far. Consider the last point in time where at least one ray has not been explored up to some depth $> 1$, and denote the point in time one time unit earlier by $t_0$. It follows that at time $t_0$ there must

be some robot $R_i$ at the junction or on a ray which at time $t_0 + 1$ has not been explored up to some depth $> 1$.

Let $\varepsilon > 0$. Let $P$ be the set of points in distance $t_0/(2x)+\varepsilon$ from the junction and observe that $t_0/(2x) \geq 1$ since $t_0 \geq 2\lfloor (m-1)/k \rfloor$. We claim that at time $t_0 + t_0/(2x)$, robot $R_i$ has explored at most $x - 1$ points in $P$: Since $R_i$ starts in the junction and, at time $t_0$, is again in the junction or on a ray where the corresponding point from $P$ will not be explored up to and including time $t_0+1$, it must travel a total distance of at least $2y(t_0/(2x) + \varepsilon)$ in order to explore $y$ points from $P$ up to time $t_0$. Thus, we obtain $2y(t_0/(2x)+\varepsilon) \leq t_0$ which implies $y < x$ and thereby proves the claim. Note that robot $R_i$ cannot explore a point from $P$ between $t_0$ and $t_0 + t_0/(2x)$ because of its location at time $t_0$.

Moreover, we claim that at time $t_0 + t_0/(2x)$, any robot $R_h$ with $h \neq i$ has explored at most $x$ points in $P$: Similarly to above, in order to explore $y$ points from $P$ starting in the junction, robot $R_h$ has to travel a distance of at least $(2y-1)(t_0/(2x)+\varepsilon)$. We obtain $(2y-1)(t_0/(2x)+\varepsilon) \leq t_0+t_0/(2x)$ which implies $y < x + 1$ and thereby proves the claim. Hence, at time $t_0 + t_0/(2x)$, at most $kx - 1 \leq m - 2$ points from $P$ have been explored in total. Thus, there exist two points $p_1, p_2 \in P$ that have not been explored at time $t_0 + t_0/(2x)$. Let $t_1$ and $t_2$ be the points in time when $p_1$ and $p_2$ are explored (for the first time), respectively. W.l.o.g. assume that $t_1 \leq t_2$.

Now consider the input instance where the exit is at point $p_2$. Since some robot is at $p_1$ at time $t_1 \geq t_0 + t_0/(2x)$, this robot cannot be at $p_2$ before time $t_1 + 2(t_0/(2x) + \varepsilon)$, by the definition of $P$. We obtain a lower bound of

$$\frac{t_0 + \frac{t_0}{2x} + 2(\frac{t_0}{2x} + \varepsilon)}{\frac{t_0}{2x} + \varepsilon} = 2 + \frac{2x+1}{1 + \frac{2\varepsilon x}{t_0}}$$

for the competitive ratio. By making $\varepsilon$ arbitrarily small our lower bound gets arbitrarily close to $3 + 2x = 3 + 2\lfloor (m-1)/k \rfloor$ which proves the theorem statement. □

## 5 Concluding Remarks

We studied the problem of collaboratively evacuating $k$ robots on $m$ concurrent rays, using wireless communication. To the best of our knowledge, our work is the first that considers not starting on the junction $j$ of the $m$ rays, and also to consider $k < m$ robots for the specific problem of wireless collaborative evacuation on $m$ rays.

For the case of $k = m$ robots, a simple waiting strategy gives a competitive ratio of 4, with a constructive lower bound of $2 + \sqrt{7/3} \approx 3.52753$ for every $k = m \geq 3$. For the specific case of $k = m = 3$, we develop a parametrized class of algorithms with a nearly matching competitive ratio of $2 + \sqrt{3} \approx 3.73205$, where the parameter choice decides on the first search depth beyond the junction $j$, once the junction is found.

Unlike prior work, not starting on the junction $j$ allows to consider the scenario of the robots being initially oblivious to the number of rays. Our optimization over the parameter space of a geometric search strategy yields an algorithm with a competitive ratio of $1 + 2 \cdot \frac{m-1}{k} \cdot \left(1 + \frac{k}{m-1}\right)^{1+\frac{m-1}{k}}$. For a lower bound, we give a simple but fundamental argument, resulting in the fact that no algorithm can obtain a better competitive ratio than $3 + 2\lfloor (m-1)/k \rfloor$ for every combination of $k, m$ with $k < m$ – even when starting on $j$.

**Acknowledgments.** We would like to thank the anonymous reviewers for their helpful comments. Klaus-Tycho Foerster is supported by the Danish VILLUM FONDEN project ReNet.

## References

1. Angelopoulos, S.: Deterministic searching on the line. In: Kao, M.-Y. (ed.) Encyclopedia of Algorithms, pp. 531–533. Springer, New York (2016). https://doi.org/10.1007/978-3-642-27848-8_106-2
2. Angelopoulos, S., Arsénio, D., Dürr, C., López-Ortiz, A.: Multi-processor search and scheduling problems with setup cost. Theory Comput. Syst. **60**, 1–34 (2016)
3. Angelopoulos, S., López-Ortiz, A., Panagiotou, K.: Multi-target ray searching problems. Theor. Comput. Sci. **540**, 2–12 (2014)
4. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **106**(2), 234–252 (1993)
5. Baeza-Yates, R.A., Schott, R.: Parallel searching in the plane. Comput. Geom. **5**, 143–154 (1995)
6. Beck, A.: On the linear search problem. Israel J. Math. **2**(4), 221–228 (1964)
7. Beck, A., Newman, D.J.: Yet more on the linear search problem. Israel J. Math. **8**(4), 419–429 (1970)
8. Bellman, R.: An optimal search. SIAM Rev. **5**(3), 274 (1963)
9. Bose, P., De Carufel, J.-L.: A general framework for searching on a line. In: Kaykobad, M., Petreschi, R. (eds.) WALCOM 2016. LNCS, vol. 9627, pp. 143–153. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30139-6_12
10. Bose, P., De Carufel, J.-L., Durocher, S.: Searching on a line: a complete characterization of the optimal solution. Theoret. Comput. Sci. **569**, 24–42 (2015)
11. Brandt, S., Laufenberg, F., Lv, Y., Stolz, D., Wattenhofer, R.: Collaboration without communication: evacuating two robots from a disk. In: Fotakis, D., Pagourtzis, A., Paschos, V.T. (eds.) CIAC 2017. LNCS, vol. 10236, pp. 104–115. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57586-5_10
12. Chrobak, M., Gąsieniec, L., Gorry, T., Martin, R.: Group search on the line. In: Italiano, G.F., Margaria-Steffen, T., Pokorný, J., Quisquater, J.-J., Wattenhofer, R. (eds.) SOFSEM 2015. LNCS, vol. 8939, pp. 164–176. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46078-8_14
13. Czyzowicz, J., Dobrev, S., Georgiou, K., Kranakis, E., MacQuarrie, F.: Evacuating two robots from multiple unknown exits in a circle. In: Proceedings of 17th International Conference on Distributed Computing and Networking, Singapore, 4–7 January 2016, pp. 28:1–28:8. ACM (2016)

14. Czyzowicz, J., Gąsieniec, L., Gorry, T., Kranakis, E., Martin, R., Pajak, D.: Evacuating robots via unknown exit in a disk. In: Kuhn, F. (ed.) DISC 2014. LNCS, vol. 8784, pp. 122–136. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45174-8_9
15. Czyzowicz, J., Georgiou, K., Kranakis, E., Narayanan, L., Opatrny, J., Vogtenhuber, B.: Evacuating robots from a disk using face-to-face communication (extended abstract). In: Paschos, V.T., Widmayer, P. (eds.) CIAC 2015. LNCS, vol. 9079, pp. 140–152. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18173-8_10
16. Czyzowicz, J., Godon, M., Kranakis, E., Godon, M., Krizanc, D., Rytter, W., Wlodarczyk, M.: Evacuation from a disc in the presence of a faulty robot. In: Proceedings of SIROCCO (2017)
17. Czyzowicz, J., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J.: Search on a line with faulty robots. In: Giakkoupis, G. (ed.) Proceedings of 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, 25–28 July 2016, pp. 405–414. ACM (2016)
18. Czyzowicz, J., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J., Shende, S.: Wireless autonomous robot evacuation from equilateral triangles and squares. In: Papavassiliou, S., Ruehrup, S. (eds.) ADHOC-NOW 2015. LNCS, vol. 9143, pp. 181–194. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19662-6_13
19. Demaine, E.D., Fekete, S.P., Gal, S.: Online searching with turn cost. Theor. Comput. Sci. **361**(2–3), 342–355 (2006)
20. Foerster, K.-T., Wattenhofer, R.: Lower and upper competitive bounds for online directed graph exploration. Theor. Comput. Sci. **655**, 15–29 (2016)
21. Gal, S.: Minimax solutions for linear search problems. SIAM J. Appl. Math. **27**(1), 17–30 (1974)
22. Hammar, M., Nilsson, B.J., Schuierer, S.: Parallel searching on m rays. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 132–142. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49116-3_12
23. Hammar, M., Nilsson, B.J., Schuierer, S.: Parallel searching on m rays. Comput. Geom. **18**(3), 125–139 (2001)
24. Jaillet, P., Stafford, M.: Online searching. Oper. Res. **49**(4), 501–515 (2001)
25. Kamphans, T., Langetepe, E.: Optimal competitive online ray search with an error-prone robot. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 593–596. Springer, Heidelberg (2005). https://doi.org/10.1007/11427186_51
26. Kao, M.-Y., Ma, Y., Sipser, M., Yin, Y.L.: Optimal constructions of hybrid algorithms. J. Algorithms **29**(1), 142–164 (1998)
27. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. Inf. Comput. **131**(1), 63–79 (1996)
28. Komm, D., Královič, R., Královič, R., Smula, J.: Treasure hunt with advice. In: Scheideler, C. (ed.) Structural Information and Communication Complexity. LNCS, vol. 9439, pp. 328–341. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25258-2_23
29. López-Ortiz, A., Schuierer, S.: The ultimate strategy to search on m rays? Theoret. Comput. Sci. **261**(2), 267–295 (2001)
30. López-Ortiz, A., Schuierer, S.: On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. Theoret. Comput. Sci. **310**(1–3), 527–537 (2004)

31. Schuierer, S.: A lower bound for randomized searching on m rays. In: Klein, R., Six, H.-W., Wegner, L. (eds.) Computer Science in Perspective. LNCS, vol. 2598, pp. 264–277. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36477-3_20
32. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
33. Tate, S.R.: Randomized searching on rays or the line. In: Kao, M.-Y. (ed.) Encyclopedia of Algorithms, pp. 1757–1759. Springer, New York (2016). https://doi.org/10.1007/978-0-387-30162-4_328

# Evacuation from a Disc in the Presence of a Faulty Robot

Jurek Czyzowicz[1], Konstantinos Georgiou[2], Maxime Godon[1], Evangelos Kranakis[3(✉)], Danny Krizanc[4], Wojciech Rytter[5], and Michał Włodarczyk[5]

[1] Departement d'Informatique, Université du Québec en Outaouais, Gatineau, Canada
[2] Department of Mathematics, Ryerson University, Toronto, ON, Canada
[3] School of Computer Science, Carleton University, Ottawa, ON, Canada
kranakis@cs.carleton.ca
[4] Department of Mathematics, Wesleyan University, Middletown, CT, USA
[5] Institute of Informatics, University of Warsaw, Warsaw, Poland

**Abstract.** We consider the evacuation problem on a circle for three robots, at most one of which is faulty. The three robots starting from the center of a unit circle search for an exit placed at an unknown location on the perimeter (of the circle). During the search, robots can communicate wirelessly at any distance. The goal is to minimize the time that the latest non-faulty robot reaches the exit.

Our main contributions are two intuitive evacuation protocols for the non-faulty robots to reach the exit in two well-studied fault-models, Crash and Byzantine. Moreover, we complement our positive results by lower bounds in both models. A summary of our results reads as follows:

- *Case of Crash Faults:*
  Lower Bound ≈5.188; Upper Bound ≈6.309,
- *Case of Byzantine Faults:*
  Lower Bound ≈5.948; Upper Bound ≈6.921,

For comparison, it is known (see [11]) that in the case of three *non-faulty* robots with wireless communication we have a lower bound of 4.159, and an upper bound of 4.219 for evacuation, while for two non-faulty robots $1 + 2\pi/3 + \sqrt{3} \approx 4.779$ is a tight upper and lower bound for evacuation.

**Keywords:** Algorithm · Byzantine faulty · Crash faulty · Evacuation Robot · Search

J. Czyzowicz, K. Georgiou, and E. Kranakis—Research supported in part by NSERC Discovery grant.
J. Czyzowicz and W. Rytter—Supported in part by grant NCN2014/13/B/ST6/00770 of the Polish Science Centre.
M. Włodarczyk—Supported in part by the National Science Centre of Poland Grant UMO-2016/21/N/ST6/00968.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 158–173, 2017.
https://doi.org/10.1007/978-3-319-72050-0_10

# 1 Introduction

Searching an environment to find an exit (or target) placed at an unknown location has been studied extensively in computer science and robotics. The searchers are autonomous robots which (may) cooperate during their search by exchanging messages so that at least one of them can find the target in minimum possible time. Another form of search recently introduced in [11] is called *evacuation* and it has the additional requirement that all the robots must go to the exit. Thus, optimality in evacuation is measured by the time it takes for the last robot to reach the exit, whereas in traditional search, optimality is measured by the time it takes the first robot to reach the exit.

In this paper we consider an evacuation problem for three robots which are able to communicate wirelessly. Initially, the robots are located at the center of a disc of radius one and must find an exit located on the circumference of the disc and then gather at the location of the exit. We consider two scenarios in which exactly one robot is faulty. In the first scenario, one robot can experience crash faults, which prevents it from either communicating or locating the exit. In the second scenario, one robot can experience Byzantine faults, which allow it to lie, e.g., to claim to have found an exit–where there is none–or even to fail to report (communicate) the location of the exit to the other robots. Note that the evacuation problem is considered to be solved when both non-faulty robots find the exit. For both scenarios, we provide upper and lower bounds.

## 1.1 Preliminaries/The Model

There are three robots initially located at the center of a unit disc. The robots can move with maximum speed 1 (thus, they may stop or change direction at no cost), and are required to find an exit (whose location is unknown to the robots) located somewhere on the circumference of the disc and then gather at this location as fast as possible. On the perimeter of the disc the robots have a sense of direction and can distinguish between clockwise and counterclockwise direction of movement. A robot can find the exit only when it is in the same location as the exit. During their search the robots employ a *wireless communication model*, which means that they can exchange information instantaneously and at no cost and at any time, no matter the distance that separates them during their search.

The search problem to be studied is concerned with all non-faulty robots evacuating from the (unknown) exit. The search task is complicated by the fact that one of the three robots, chosen by an adversary, experiences faults, chosen by the adversary as well. We consider two scenarios. In the first scenario, the faulty robot experiences *crash* faults while in the second the robot experiences *Byzantine* faults. In both cases, the goal is to minimize the time till the last non-faulty robot reaches the exit.

– Crash-Evacuation: A *crash* fault can be thought of as a passive fault rending: a robot is either unable or incapable to either detect or report the exit when it reaches it. Thus, such a robot is not expected to find the exit, only

non-faulty robots can. However, we assume that in other aspects, a faulty robot moves like a non-faulty robot, and thus non-faulty robots cannot detect which robots are faulty.
– Byzantine-Evacuation: A *Byzantine* faulty robot not only can fail to detect or report the target even after reaching it, it can also make malicious claims about having found the target when in fact it has not. Given the presence of such a faulty robot, the search for the target can only be concluded when the two non-faulty robots have sufficient verification that the target has been found.

All the messages being transmitted by the robots are tagged with the robot's unique identifier, which cannot be altered.

### 1.2 Related Work

Searching an environment to find an exit placed at an unknown location is a well studied problem in computer science and robotics. The searchers are autonomous mobile robots that may also possess partial knowledge of their environment. Many researchers, starting with the seminal work of Bellman [5] and Beck [4], have studied the optimal (length) trajectory traced by a single robot when searching for a target placed at an unknown location on a line. The aim of the algorithmic designer is to minimize the competitive ratio, that is, the supremum, over all possible target locations, of the ratio between the distance traveled by the robot until it finds the exit, and the distance of the exit from the robot's starting position. For the case of a single robot on a line, the optimal trajectory uses a zig-zag, doubling strategy according to which if the robot fails to find the exit after travelling a certain distance in a particular direction it returns to its starting position and doubles its searching distance in the opposite direction. This trajectory has a competitive ratio of 9 and this can be shown to be optimal (e.g., see Baeza-Yates and Schott [3]).

Several authors considered the problem of searching in the two-dimensional plane by one or more searchers, including [2,3]. The evacuation problem on a unit disc for multiple robots considered in our present work is a form of two-dimensional search that was first considered in [11]. In that paper the authors studied evacuation algorithms in the wireless and face-to-face communication models. New algorithms for the face-to-face communication model were subsequently analyzed for two robots in [14] and later in [7]. The problem has also been considered in other domains, like triangles and squares in [16]. However, all these papers concern evacuation only for non-faulty robots.

One of the novelties of our current work is that we consider the two-dimensional evacuation problem with fault tolerance. There are numerous studies of fault tolerance in distributed computing, (see, e.g., [19,22,23]). Network failures were most frequently related to static elements of the networked environment (i.e., nodes and links) as opposed to its mobile components. Malfunctions of this kind were sometimes modelled by dynamic alteration of the network [8,21]. Distributed computation arising when having some of the mobile robots are faulty were investigated in the context of the problems of gathering [1,17,18,24],

convergence [6,9], flocking [25], and patrolling [12]. Several researchers also investigated the case of unreliable or inaccurate robot sensing devices, e.g., [10,20,24]. Related to our study is also the research in [12], where a collection of robots, some of which are unreliable, perform efficient patrolling of a fence. Most relevant to our current study for its perspective on search and fault tolerance is the research of [13,15] which propose search algorithms for faulty robots that may suffer from crash and Byzantine faults, respectively.

### 1.3 Outline and Results of the Paper

An outline of this paper can be described as follows. Section 2 is dedicated to upper bounds. In Sects. 2.1 and 2.2 we provide evacuation protocols along with their (worst case) analyses for the Crash-Evacuation problem and the Byzantine-Evacuation problem, respectively. Then, in Sect. 3 we give lower bounds for both problems. Section 4 gives a discussion of possibilities for further research. The main results of the paper are summarized in Table 1. Notably, since the optimal offline algorithm for both problems Crash-Evacuation and Byzantine-Evacuation would have the robots move directly to the exit at time 1, the time bounds of Table 1 can be also understood as bounds for the competitive ratio of the underlying online problems.

**Table 1.** Comparison of Crash vs Byzantine: the first column gives the type of fault, the middle column lower bounds, and the right column upper bounds for the corresponding type of faults.

| Problem | Lower bound | Upper bound |
|---|---|---|
| Crash-Evacuation | $\approx$5.188 (Theorem 3) | $\approx$6.309 (Theorem 1) |
| Byzantine-Evacuation | $\approx$5.948 (Theorem 3) | $\approx$6.921 (Theorem 2) |

It is interesting to compare the results obtained in our paper to the case of non-faulty robots. It is known (see [11]) that in the case of three *non-faulty* robots with wireless communication we have a lower bound of 4.159, and an upper bound of 4.219 for evacuation, while for two non-faulty robots $1 + 2\pi/3 + \sqrt{3} \approx 4.779$ is a tight upper and lower bound for evacuation.

## 2 Evacuation Protocols

In this section we propose evacuation algorithms for crash and Byzantine faults, respectively.

### 2.1 Evacuating with Crash-Faults

The main contribution is as follows.

**Theorem 1.** Crash-Evacuation *can be solved in time* $\approx$6.309.

We prove Theorem 1 by identifying the *best* among a special family of natural algorithms that we call *persistent*. These are algorithms that force all robots to immediately go to the circumference of the disc, and only allow a robot to stop exploring its segment of the disc (either by changing direction, by becoming idle or by leaving the circumference entirely) when it receives information about the exit. Since in this model, a faulty robot can only stay silent, any report about the exit has to be valid. As such, once the location of the exit is received by a robot, the robot moves along the shortest chord toward the reported exit, and evacuates.

We further classify persistent algorithms in two categories: the *symmetric-persistent* that have all the robots begin their exploration in the same direction (either all clockwise or all counter-clockwise), and the *asymmetric-persistent* that have one robot go in a direction, and the other two robots go in the opposite direction. It turns out that the best *asymmetric-persistent* algorithm outperforms the best *symmetric-persistent* algorithm (and also proves Theorem 1). Nevertheless, and as a warm-up, we begin by providing a tight analysis for the family of *symmetric-persistent* algorithms.

**Lemma 1.** *The best* symmetric-persistent *algorithms deploys the three robots at equidistant points on the disk (at arc-distance* $4\pi/3$*), and its performance is* $1 + \frac{4\pi}{3} + \sqrt{3}$.

*Proof* (Lemma 1). Consider a symmetric-persistent algorithm that deploys robots $r_1, r_2, r_3$ so that their pairwise anti-clock-wise distance is $\beta, \gamma$ and $\alpha$ respectively, as also depicted in Fig. 1 (where also arcs $A, B, C$ are defined). Without loss of generality, assume the robots move in clockwise direction.

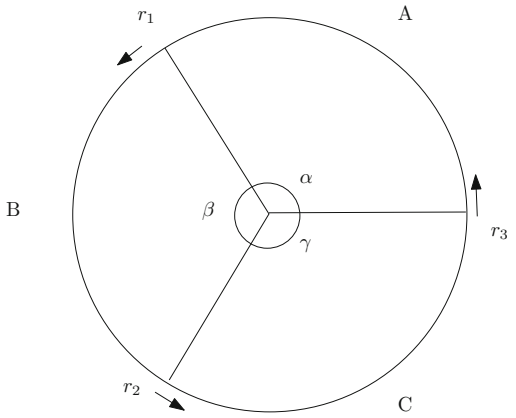



**Fig. 1.** All robots move counter-clockwise. Arc $A$ includes $r_3$ and excludes $r_1$; arc $B$ includes $r_1$ and excludes $r_2$; and arc $C$ includes $r_2$ and excludes $r_3$.

Consider the case where $r_1$ is faulty and the robots traverse the arcs depicted in Fig. 1. Clearly, there are two cases to consider depending on whether the exit

is located in one of the arcs $A$ or $B$, or the exit is located on arc $C$. If the exit is located in one of the arcs $A$ or $B$, then $r_3$ will discover it. If the exit is located in $C$, then $r_2$ will discover it. We say that the exit is either located at a counter-clockwise arc distance of $0 \le x < \gamma$ from $r_2$ if $r_2$ discovers the exit, or a counter-clockwise arc distance of $0 \le y < \alpha + \beta$ from $r_3$ if $r_3$ discovers the exit. Therefore, the total amount of time required to find the exit is given by the formula

$$1 + \max\left\{ \sup_{0\le x<\gamma} \left(x + 2\sin\frac{\gamma}{2}\right), \sup_{0\le y<\alpha+\beta} \left(y + 2\sin\frac{\alpha+\beta}{2}\right) \right\}$$
$$= 1 + \max\left\{f(\gamma), f(\alpha+\beta)\right\},$$

where we define $f(x) := x + 2\sin\frac{x}{2}$.

Similarly, if $r_2$ or $r_3$ is faulty, then the algorithm terminates in time $1 + \max\{f(\gamma), f(\beta+\gamma)\}$ and $1 + \max\{f(\beta), f(\alpha+\gamma)\}$ respectively. We conclude that the best symmetric-adaptive algorithm would choose $\alpha, \beta, \gamma$ (partitioning the perimeter of the circle, of length $2\pi$) so as to minimize quantity

$$1 + \max\{f(\alpha), f(\beta), f(\gamma), f(\alpha+\beta), f(\beta+\gamma), f(\alpha+\gamma),\} \tag{1}$$

By choosing $\alpha = \beta = \gamma = \frac{4\pi}{3}$, expression (1) gives completion time $1 + \frac{4\pi}{3} + \sqrt{3}$ as promised.

Finally, we argue that no values of $\alpha$, $\beta$ and $\gamma$ respecting $\alpha$, $\beta$ and $\gamma \ge 0$ and $\alpha + \beta + \gamma = 2\pi$ can improve on this bound. Say, we set $\alpha > \frac{2\pi}{3}$. Then it is clear that either $\alpha + \beta > \frac{4\pi}{3}$ or $\alpha + \gamma > \frac{4\pi}{3}$, since $\alpha + \beta + \gamma = 2\pi$. Observe that function $\alpha + \beta + 2\sin\frac{\alpha+\beta}{2}$ is increasing in $\alpha + \beta$, and when $\alpha + \beta = \frac{4\pi}{3}$, then (1) is upper bounded by $1 + \frac{4\pi}{3} + \sqrt{3}$. Observe also that function $\alpha + \gamma + 2\sin\frac{\alpha+\gamma}{2}$ is increasing in $\alpha + \gamma$, and when $\alpha + \gamma = \frac{4\pi}{3}$, then expression (1) is upper bounded by $1 + \frac{4\pi}{3} + \sqrt{3}$. We conclude that function (1) strictly increases for $\alpha > \frac{2\pi}{3}$. A similar argument shows that function (1) increases if either $\beta$ or $\gamma$ exceed $\frac{2\pi}{3}$. This completes the proof of Lemma 1. □

In order to proceed with the analysis of asymmetric-persistent algorithms, we need a simple technical lemma, providing a worst case analysis for a special configuration of healthy searching robots.

**Lemma 2.** *Consider two robots at arc distance $2\pi - s$ that are about to explore an arc of length $s$ moving in opposing directions (toward each other). Assume also that an exit is located somewhere at the arc of length $s$. Then, the worst case termination time $g(s)$ is given by the formula*

$$g(s) = \begin{cases} 2\sin(s/2) & , \text{ if } s < 2\pi/3 \\ s/2 - \pi/3 + \sqrt{3} & , \text{ otherwise.} \end{cases}$$

*Proof* (Lemma 2). By symmetry, we may assume that the exit is found after time $x$ by one of the robots, where $0 \le x \le s/2$ (see Fig. 2). When the message is transmitted that the exit is found, the two robots are at the endpoints of an arc

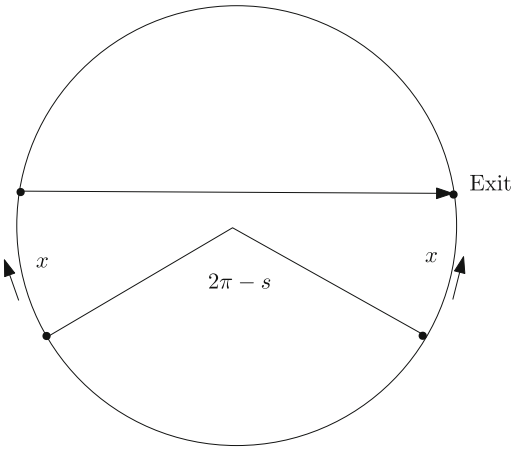



**Fig. 2.** Exit found and reported after time $x$. Worst case is $x = 0$, if $s \leq 2\pi/3$, and $x = s/2 - \pi/3$ otherwise.

of length $s - 2x$, hence at chord distance $2\sin(s/2 - x)$. Hence, the time elapsed till both robots reach the exit is $x + 2\sin(s/2 - x)$. The claim follows by the monotonicity of the latest function with respect to $x$ in the interval $[0, s/2]$. This completes the proof of Lemma 2. □

We are now ready to prove Theorem 1, by determining the optimal asymmetric-persistent algorithm.

**Lemma 3.** *The best* asymmetric-persistent *algorithm has performance* ≈6.309. *The algorithm achieving this bound deploys two robots to the same location on the disc, which they explore in opposing directions. The third robot is deployed at arc-distance $\beta_0$ from any of the robots, and starts exploring in opposite direction of the closest robot, where $\beta_0$ is the unique root of $3\beta/2 + \sqrt{3} = 4\pi/3 + 2\sin(\beta/2)$ in the interval $[0, 2\pi]$.*

*Proof* (Lemma 3). Consider an asymmetric-persistent algorithm that deploys robots $r_1, r_2, r_3$ as depicted in Fig. 3, where $\alpha, \beta > 0$ (the case $\beta = 0$ can be easily seen to induce worse termination time, while the case $\alpha = 0$ is identical to $\gamma = 0$).

There are a number of cases as to which the faulty robot is and where the exit is located. All the cases are summarized in Table 2, where identical cases are also grouped together.

For each case we will determine the worst case running time. Then we will choose $\alpha, \beta, \gamma$ so as to minimize the maximum of all these running times.

– *Case 1.* After time $\gamma$, robots $r_2, r_3$ will be at arc distance $\gamma$ and they will be about to explore an arc of length $\alpha + \beta = 2\pi - \gamma$ moving in opposing directions. Also the exit is located somewhere at the arc of length $2\pi - \gamma$.

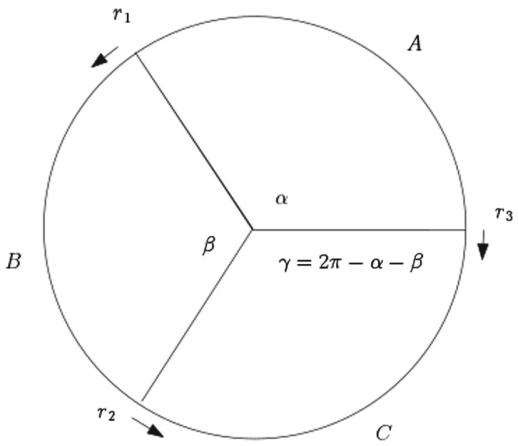



**Fig. 3.** Robots $r_1$ and $r_2$ move counter-clockwise; $r_3$ moves clockwise. $A$ excludes the starting position of $r_1$ and $r_3$; $B$ excludes the starting position of $r_2$, but includes the starting position of $r_1$; $C$ includes the starting position of both $r_2$ and $r_3$.

**Table 2.** The columns indicate the location of the exit. The rows indicate the faulty robot. $r_1$'s initial search position is in B, $r_2$ and $r_3$'s initial search position are in C.

| | $A$ | $B$ | $C$ |
|---|---|---|---|
| $r_1$ | Case 1 | Case 1 | Case 2 |
| $r_2$ | Case 3 | Case 4 | Case 4 |
| $r_3$ | Case 5 | Case 6 | Case 5 |

Hence, by Lemma 2, the (worst case) total termination time will be $1 + \gamma + g(2\pi - \gamma)$ which simplifies to

$$e(\gamma) := \begin{cases} 1 + \gamma + 2\sin(\gamma/2) & , \text{if } \gamma > 4\pi/3 \\ 1 + \gamma/2 + 2\pi/3 + \sqrt{3} & , \text{otherwise.} \end{cases}$$

Also, it is easy to see that $e(\gamma)$ is strictly increasing, a fact we will use later on.

- *Case 2.* The setup is identical to that of Lemma 2 where the arc that holds the exit has arc length $s = \gamma$. Hence, the (worst case) total termination time will be $1 + g(\gamma)$, which is easily seen to be dominated by $e(\gamma)$ of case 1, for every $0 \leq \gamma \leq 2\pi$.
- *Case 3.* This situation is similar to Case 1, where (instead of $\gamma$) robots are at distance $\beta + \gamma$, and they are moving toward each other, and in an arc segment that does not contain the exit. Hence, the worst case termination time is equal to $e(\beta + \gamma)$. Since $e(\cdot)$ is strictly increasing, this case dominates the cost of case 1.
- *Case 4.* This situation is similar to Case 2, where (instead of $\gamma$) robots are at distance $\beta + \gamma$ and they are moving toward one another and toward the segment that contains the exit. The maximal total required time is therefore

given by the function $1 + g(\beta + \gamma)$, which is easily seen to be dominated by $e(\beta + \gamma)$ of case 3, for all $0 \leq \beta + \gamma \leq 2\pi$.

- *Case 5.* We treat the case when $r_3$ is faulty and the exit is either in $C$ or $A$ together. It is clear that $r_2$ will be the robot that finds the exit. Assume that the exit is located at distance $0 \leq x < \alpha + \gamma$ from the initial searching position of $r_2$ (to ensure that the exit is located in A). Then the total required search time is given by $1 + x + 2\sin\frac{\beta}{2}$, since the distance between $r_1, r_2$ remains invariant. Clearly, in the worst case, the total required search time is $1 + \alpha + \gamma + 2\sin\frac{\beta}{2}$.
- *Case 6.* This case is identical to case 5, where $r_1$ will find the exit (instead of $r_2$, but still $\beta$ remains their invariant distance), and where the arc that contains the exit has length $\beta$ (instead of $\alpha+\gamma$). Hence, worst case termination time is equal to $1 + \beta + 2\sin\frac{\beta}{2}$.

It follows that the best asymmetric-persistent algorithm is determined by $\alpha, \beta, \gamma$ that minimize

$$\max\left\{e(\beta+\gamma), 1+\alpha+\gamma+2\sin(\beta/2), 1+\beta+2\sin(\beta/2)\right\},$$

i.e. the costs of cases 3, 5, and 6.

First we show that the promised upper bound is achievable. Indeed, we set $\gamma = 0$, so that $\alpha + \beta = 2\pi$. Now we define $\beta_0$, by equating the costs of cases 3, 5, i.e. as the root of the equation $e(\beta) = 1 + 2\pi - \beta + 2\sin(\beta/2)$. Numerical calculations yield that $\beta_0 \approx 2.96603$, or in other words (by looking at the definition of function $e(\beta)$), $\beta_0$ is defined as the solution to the equation $3\beta/2+\sqrt{3} = 4\pi/3+2\sin(\beta/2)$. We conclude that $\gamma = 2\pi-\beta_0 \approx 3.31716 < 4\pi/3$, which induces worst termination time to be the same in cases 3, 5 and equal to $1 + 2\pi - \beta_0 + 2\sin(\beta_0/2) \approx 6.30946$, as promised.

Now we prove the above choices are optimal. Indeed, if $\beta+\gamma > 4\pi/3$, then the total termination time cannot be better than the situation where cases 3, 5 induce the same cost. Equating the resulting costs, we obtain that $\beta + \gamma + 2\sin((\beta + \gamma)/2) = \alpha + \gamma + 2\sin(\alpha/2)$. Using that $\beta + \gamma = 2\pi - \alpha$, the previous equation yields $\beta - 2\sin(\beta/2) = \alpha - 2\sin(\alpha/2)$, i.e. that $\alpha = \beta$. But then $\gamma = 0$ as well. Since $\beta > 4\pi/3$, the induced cost, by case 3, is at least $1+4\pi/3+\sqrt{3} \approx 6.92084$.

Finally, assume that $\beta + \gamma \leq 4\pi/3$. For any fixed $\gamma$, the total termination time cannot be better than the situation where cases 3, 5 induce the same cost. Equating the resulting costs, we obtain that $(\beta + \gamma)/2 + 2\pi/3 + \sqrt{3} = \alpha + \gamma + 2\sin(\beta/2)$. Since $\alpha = 2\pi - \beta - \gamma$, the optimal choice for $\beta$ should be $\beta_\gamma$ satisfying $3\beta_\gamma/2 + \gamma/2 + \sqrt{3} = 4\pi/3 + 2\sin(\beta_\gamma/2)$. Note that $\beta_\gamma$ is a function of $\gamma$, hence differentiating both sides of last equation with respect to $\gamma$, and after elementary calculations, we obtain that $\beta'_\gamma(3/2-\cos(\beta_\gamma/2)) = -1/2$. Since $\beta_\gamma > 0$, we obtain that $\cos(\beta_\gamma/2) < 1$ and hence $\beta'_\gamma > -1$. This implies that expression $\beta_\gamma + \gamma$ is strictly increasing in $\gamma$, and this linear term appear in the termination time of case 3. Hence, choosing $\gamma = 0$ is indeed optimal. This concludes the proof of Lemma 3. □

### 2.2 Evacuating in the Presence of Byzantine Faults

The main contribution is as follows.

**Theorem 2.** BYZANTINE-EVACUATION *can be solved in time* $1 + \frac{4\pi}{3} + \sqrt{3} \approx 6.92084$.

*Proof* (Theorem 2). The analysis relies on Fig. 4. Assume that all three robots $r_k$, for $k \in \{1, 2, 3\}$, execute the main evacuation Algorithm 1.

The idea of the algorithm is for the robots to traverse the circumference of the disk for a time of $2\pi/3$. Depending on the calls that have been received, the robots have information to either go to the exit or continue traversing the circumference of the disk for another period of time $\frac{2\pi}{3}$. They can now verify conflicting messages of the correct location of the exit based on the calls that have been made by the other robots so far. Details are being discussed in the sequel.

**Algorithm 1.** Evacuation with Byzantine Faults

```
1  Go to the circumference, at position 2πk/3;
2  while r_k 's location is not the same as the exit's location do
3      for 2π/3 do
4          follow the circumference clockwise
5      if One robot claims to have found more than one exit then
6          Continue execution of algorithm as though the robot remained silent
7      if No information about exit then
8          for 2π/3 do
9              follow the circumference clockwise till exit is either found or reported.
               Finish
10     if One robot claims to have found the exit then
11         Go to closest part of the segment that is claimed to contain the exit;
12         Explore entire segment. Finish.
13     if Two robots claim to have found the exit then
14         Investigate both exits. Finish.
15 Inform all robots of the location of the exit.
```

First note that one time unit is required to reach the circumference of the disc. After $\frac{2\pi}{3}$ additional time units, the entire disc has been explored once. The areas explored by the robots are contiguous but not overlapping. Observe that a Byzantine robot that claims to have found more than one exit is immediately identified as faulty by the healthy robots. Both potential exits are ignored, and the algorithm continues as though the robot had remained silent. If a non-faulty robot finds the exit, it immediately informs all other robots, then stop its exploration. Say without loss of generality that $r_1$ is healthy. If $r_1$ finds the exit during the first $\frac{2\pi}{3}$ part of the exploration, then it stops and is done with the execution of its algorithm, in a time at most $1 + \frac{2\pi}{3}$. If it does not find an exit during the first $\frac{2\pi}{3}$ part of the exploration, then we must consider three cases:

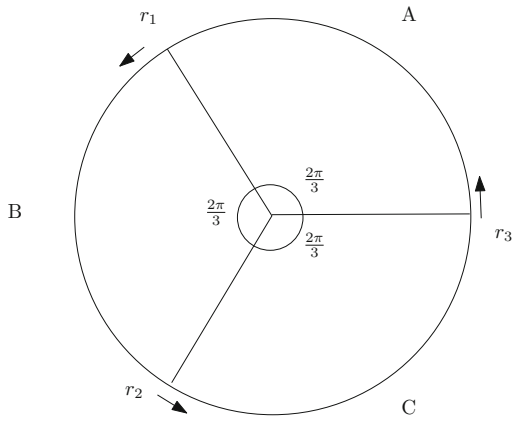



**Fig. 4.** The initial searching position for $r_1$, $r_2$ and $r_3$ in the Byzantine faults model

- *No exit location reported:* If no exit was found, then keep exploring the circumference of the disk for time $\frac{2\pi}{3}$. Notice that this means that the exit cannot be in B. If the exit is in C, then $r_1$ has found the exit, and its execution is complete in a time at most $1 + \frac{4\pi}{3}$. If the exit is in A, then we learn that $r_3$ is Byzantine (otherwise, it would have claimed to have found the exit during the first $1 + \frac{2\pi}{3}$ of the execution of the algorithm), and $r_2$ will have correctly identified the location of the exit (Notice that $r_1$ needs to finish exploring the second arc $C$ to make sure that it was r3 that lied.) Say the exit is located at an arc distance of $0 < x < \frac{2\pi}{3}$ from $r_1$'s current position. Then $2\sin\frac{x}{2}$ is required for $r_1$ to reach the exit. Since this function is monotone in $x$ for $x \leq \pi$, $r_1$ can reach the exit in a total time of at most $1 + \frac{4\pi}{3} + \sqrt{3}$.
- *One exit location reported:* If one robot other than $r_1$ claims to have found the exit, we consider two situations: (1) the robot is healthy, in which case the exit is indeed located on the segment where the announcement was made; or (2) the robot is Byzantine, in which case the other two segments have been entirely explored by healthy robots (and are therefore reliably proven to be empty), and the exit is located on the segment where the announcement was made. Notice that in both situations, the only possible location for the exit is on the segment where the announcement was made. If the announcement was made on the segment $C$, then $r_1$ explores $C$ immediately, for a total time of at most $1 + \frac{4\pi}{3}$. If the announcement was made on the segment $A$, then $r_1$ must first reach one end of segment $A$, which requires $2\sin\frac{2\pi}{3} = \sqrt{3}$ (both ends of the segment are equidistant from $r_1$'s position), then explore the segment, for a total time of at most $1 + \frac{4\pi}{3} + \sqrt{3}$.
- *Two exit locations reported:* If both $r_2$ and $r_3$ claim to have found an exit, then we know that one of those two claims is true. $r_1$ will investigate both claims, starting by the closest one. Say $r_2$ claims to have found the exit at a distance $x$ from its initial searching position, and $r_3$ claims to have found the exit at a distance $y$ of its initial searching position. Then $r_1$ must travel an additional

$2\sin\frac{x}{2} + 2\sin\frac{\frac{2\pi}{3}-x+y}{2}$ to reach both exits. This function is maximised for $x = y = \frac{2\pi}{3}$, for a total time of at most $1 + \frac{2\pi}{3} + 2\sqrt{3}$.

Observe that both robots $r_2$ and $r_3$ execute the same algorithm, and the maximal time required is therefore the same. The adversary will choose the location of the exit and the Byzantine robot in such way as to maximise the total time of execution of the algorithm. Therefore, since $\sqrt{3} < \frac{2\pi}{3}$, this algorithm solves the evacuation problem in total time $1+\frac{4\pi}{3}+\sqrt{3}$. This completes the proof of Theorem 2. □

## 3 Lower Bounds for Evacuation Protocols

This section is devoted to proving our main negative results.

**Theorem 3.** *The following lower bounds are valid.*

*(a) Problem* CRASH-EVACUATION *requires time at least* $1 + 4\pi/3 \approx 5.188$.
*(b) Problem* BYZANTINE-EVACUATION *requires time at least* 5.948.

*Proof* (Theorem 3). We separate the proofs of the two parts of Theorem 3 on crash and Byzantine faults, respectively.

*Lower Bound for Crash-Faults.* The lower bound proof for Crash faults is simple. Every point on the perimeter must be visited by at least two robots (if not, the adversary will place the exit at that point and make the robot visiting it crash faulty). Since there are three robots at least one of the robots will take time at least $4\pi/3$ after visiting the perimeter. This proves the lower bound.

*Lower Bound for Byzantine-Faults.* This case is harder to analyze and we will first consider a few preliminary lemmata below.

It is easy to observe that if we consider three robots starting from the center of a unit disc then for any $\epsilon > 0$, at time $1+\frac{2\pi}{3}-\epsilon$ there is an equilateral triangle inscribed in the circle not all of whose vertices have been explored by a robot. However, in the main proof we will make use of an even stronger property of the three robots.

Next we define a useful property $P(T)$, where $T > 0$ denotes time, to be used in the rest of the proof for a lower bound.

**Definition 1 (Property $P(T)$).** *For any algorithm and any time less than $T$ there are two points on the circle at distance at least $\sqrt{3}$ and each of which was visited at most once by anyone of the three robots.*

Since Property $P(T)$ ensures the existence of two points at distance at least $\sqrt{3}$ which have been visited at most once by the robots, a simple adversarial argument will guarantee that $T+\sqrt{3}$ is a lower bound on evacuation for Byzantine faults (see Lemma 5). However, before proving these last statements, we are interested to find a $T$ which satisfies property $P(T)$.

Note that property $P(T)$ is monotone increasing in $T$, in that $P(T) \wedge T' \leq T \Rightarrow P(T')$. Hence, the larger the value of the parameter $T$ for which $P(T)$ is valid the better the lower bound that can be derived.

**Lemma 4.** *Property* $P(1+13\sqrt{3}/7)$ *is valid.*

*Proof* (Lemma 4). In the sequel, to help our intuition, we prove first the weaker statement that $P(4)$ is valid and then we improve this to $P(1+13\sqrt{3}/7)$. Let us consider some algorithm at time $< T$, where $T = 4$, and assume by contradiction that all points that have been visited at most once by a robot are at distance less than $\sqrt{3}$ from each other. Clearly, all these points must lie on an arc of length less than $2\pi/3$. Therefore looking at the complement of this arc we find an arc of length longer than $4\pi/3$. In turn, this gives rise to a regular hexagon with five of its vertices inside this last arc each visited twice by a robot. Therefore these five vertices of the hexagon have been visited ten times in total by the three robots. Since there are three robots, it follows that at least one robot must have visited four of these vertices. However this is impossible as $T = 4$. It follows that property $P(4)$ is valid.

Now we derive the main result of the lemma by showing that $P(1+13\sqrt{3}/7)$ is valid. We argue as in the previous paragraph, however, instead of selecting five vertices of a regular hexagon we will choose the five points more carefully.

As in the proof of $P(4)$ above, let three points $A, B, C$ be vertices of an equilateral triangle such that every point in the perimeter of the disc which is visited by at most one of the three robots is in the arc clockwise between $A$ and $B$ (Fig. 5).

In turn, this will give rise to five points on the circumference of the disc with each of its vertices visited twice by a robot; namely choose a point $D$ located between $A$ and $C$ and a point $E$ between $B$ and $C$ so that the length of arc $AD$ is $x$ and this is equal to the length of arc $EB$ (the choice of $x$ will be based on maximizing the length of a path visiting these vertices and will be made precise

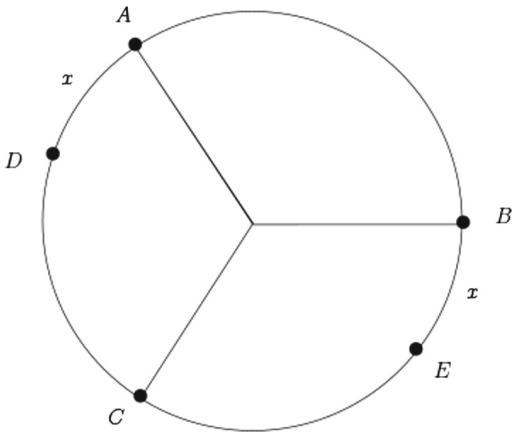



**Fig. 5.** Evacuation of the second truth telling robot.

in the next paragraph). Since there are ten visitations by three robots one of the robots must have visited four consecutive points at least once.

We will show that visiting four vertices among $A, B, C, D, E$ takes time at least $13\sqrt{3}/7 \approx 3.21$. If $x < \pi/3$ then there are 2 candidates for the shortest four-point walk, namely

$$\text{either } D \to A \to B \to E \text{ or } A \to D \to C \to E.$$

Taking into account the lengths of the corresponding chords in these two paths, it turns out that we need to maximize the function $f(x)$ defined by the equation below.

$$f(x) := \min\{\sqrt{3} + 4\sin(x/2), 2\sin(x/2) + 4\sin(\pi/3 - x/2)\}.$$

It is easily seen that the maximum of $f$ is equal to $1 + 13\sqrt{3}/7$ and it is obtained at $x = 4/\arctan(1/(3\sqrt{3}))$. The rest of the reasoning is the same as for $T = 4$ in the first paragraph of the proof. This completes the proof of Lemma 4. □

Now we can conclude the proof of Part (b) which follows as a corollary of Lemma 5 below.

**Lemma 5.** *If property $P(T)$ holds then we can achieve a lower bound of $T + \sqrt{3}$ on evacuation in the presence of a Byzantine robot.*

*Proof* (Lemma 5). Identify two points $A$, $B$ at distance $\geq \sqrt{3}$ each of which was visited at most once by anyone of the three robots. Assume without loss of generality that $r_1$ visited $A$. Then we have two possibilities to consider: either $r_1$ also visited $B$, or (say) $r_2$ visited $B$.

If $r_1$ visited both points, set $r_1$ to be Byzantine, then wait until either $r_2$ or $r_3$ visit either $A$ or $B$. Once this first visit happens, claim that the exit is located at the other point. The robot that visited the first point will require at least $\sqrt{3}$ to reach the other point, which proves the lemma in this case.

If, say, $r_2$ visited point $B$, then have $r_1$ claim that the exit is located at point $B$, and $r_2$ claim that the exit is located at point $A$ (which will happen as soon as the robots reach those points). Then $r_3$ will have to visit both points to find the real exit, since it has no means of distinguishing the reliable robot from the Byzantine robot. Choose the first point visited by robot $r_3$ not to have the exit, and set the exit at the location of the other point. Then $r_3$ requires at least $\sqrt{3}$ to reach the other point, which proves the lemma in this case as well.

Combining these two cases, this completes the proof of Lemma 5. □

If we note the following approximations for the quantities arising in Lemma 4: $1 + 13\sqrt{3}/7 \approx 4.21$ and $4/\arctan(1/(3\sqrt{3})) \approx 0.76$, then the proof of Theorem 3 is complete. □

## 4 Discussion and Open Problems

In this paper we considered the evacuation problem on a disc for three robots exactly one of which has either crash or Byzantine faults. We analyzed the

problem in both fault scenarios and gave lower bounds as well as evacuation algorithms resulting in upper bounds. There are several challenging open problems. In addition to closing the gaps between the upper and lower bounds for either robot fault (either crash or Byzantine) model with wireless communication presented in our paper, it would be interesting to investigate the evacuation problem

(a) for other types of communication models (e.g., face-to-face, or even limited visibility),
(b) for $n$ robots $f$ of which may be faulty (either crash or Byzantine), for some $1 \leq f \leq n-2$, and derive asymptotic bounds similar to the results of [11], and
(c) for robots with not necessarily identical maximum speeds.

Despite the fact that obtaining tight bounds for evacuation problems are known often to lead to functions which can be a challenge to optimize, the algorithmic insights derived by this interaction between robot mobility and communication can lead to rewarding applications of distributed computing in search and evacuation.

## References

1. Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots. SIAM J. Comput. **36**(1), 56–82 (2006)
2. Baeza-Yates, R., Culberson, J., Rawlins, G.: Searching in the plane. Inf. Comput. **106**(2), 234–252 (1993)
3. Baeza-Yates, R., Schott, R.: Parallel searching in the plane. Comput. Geom. **5**(3), 143–154 (1995)
4. Beck, A.: On the linear search problem. Israel J. Math. **2**(4), 221–228 (1964)
5. Bellman, R.: An optimal search. SIAM Rev. **5**(3), 274–274 (1963)
6. Bouzid, Z., Potop-Butucaru, M.G., Tixeuil, S.: Optimal byzantine-rezilient convergence in uni-dimensional robot network. Theoret. Comput. Sci. **411**(34–36), 3154–3168 (2010)
7. Brandt, S., Laufenberg, F., Lv, Y., Stolz, D., Wattenhofer, R.: Collaboration without communication: evacuating two robots from a disk. In: Fotakis, D., Pagourtzis, A., Paschos, V.T. (eds.) CIAC 2017. LNCS, vol. 10236, pp. 104–115. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57586-5_10
8. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. In: Frey, H., Li, X., Ruehrup, S. (eds.) ADHOC-NOW 2011. LNCS, vol. 6811, pp. 346–359. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22450-8_27
9. Cohen, R., Peleg, D.: Convergence properties of the gravitational algorithm in asynchronous robot systems. SIAM J. Comput. **41**(1), 1516–1528 (2005)
10. Cohen, R., Peleg, D.: Convergence of autonomous mobile robots with inaccurate sensors and movements. SIAM J. Comput. **38**(1), 276–302 (2008)
11. Czyzowicz, J., Gąsieniec, L., Gorry, T., Kranakis, E., Martin, R., Pajak, D.: Evacuating robots via unknown exit in a disk. In: Kuhn, F. (ed.) DISC 2014. LNCS, vol. 8784, pp. 122–136. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45174-8_9

12. Czyzowicz, J., Gasieniec, L., Kosowski, A., Kranakis, E., Krizanc, D., Taleb, N.: When patrolmen become corrupted: monitoring a graph using faulty mobile robots. In: Elbassioni, K., Makino, K. (eds.) ISAAC 2015. LNCS, vol. 9472, pp. 343–354. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48971-0_30
13. Czyzowicz, J., Georgiou, K., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J., Shende, S.: Search on a line by byzantine robots. In: 27th International Symposium on Algorithms and Computation, ISAAC 2016, 12–14 December 2016, Sydney, Australia, pp. 27:1–27:12 (2016)
14. Czyzowicz, J., Georgiou, K., Kranakis, E., Narayanan, L., Opatrny, J., Vogtenhuber, B.: Evacuating robots from a disk using face-to-face communication (extended abstract). In: Paschos, V.T., Widmayer, P. (eds.) CIAC 2015. LNCS, vol. 9079, pp. 140–152. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18173-8_10
15. Czyzowicz, J., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J.: Search on a line with faulty robots. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25–28, pp. 405–414 (2016)
16. Czyzowicz, J., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J., Shende, S.: Wireless autonomous robot evacuation from equilateral triangles and squares. In: Papavassiliou, S., Ruehrup, S. (eds.) ADHOC-NOW 2015. LNCS, vol. 9143, pp. 181–194. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19662-6_13
17. Défago, X., Gradinariu, M., Messika, S., Raipin-Parvédy, P.: Fault-tolerant and self-stabilizing mobile robots gathering. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 46–60. Springer, Heidelberg (2006). https://doi.org/10.1007/11864219_4
18. Dieudonné, Y., Pelc, A., Peleg, D.: Gathering despite mischief. ACM Trans. Algorithms (TALG) **11**(1), 1 (2014)
19. Hromkovič, J., Klasing, R., Monien, B., Peine, R.: Dissemination of information in interconnection networks (broadcasting & gossiping). In: Du, D.Z., Hsu, D.F. (eds.) Combinatorial Network Theory, pp. 125–212. Springer, Boston (1996). https://doi.org/10.1007/978-1-4757-2491-2_5
20. Izumi, T., Souissi, S., Katayama, Y., Inuzuka, N., Défago, X., Wada, K., Yamashita, M.: The gathering problem for two oblivious robots with unreliable compasses. SIAM J. Comput. **41**(1), 26–46 (2012)
21. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: Proceedings of the Forty-second ACM Symposium on Theory of Computing, pp. 513–522. ACM (2010)
22. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. (TOPLAS) **4**(3), 382–401 (1982)
23. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, Burlington (1996)
24. Souissi, S., Défago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 333–349. Springer, Heidelberg (2006). https://doi.org/10.1007/11945529_24
25. Yang, Y., Souissi, S., Défago, X., Takizawa, M.: Fault-tolerant flocking for a group of autonomous mobile robots. J. Syst. Softw. **84**(1), 29–36 (2011)

# On Location Hiding in Distributed Systems

Karol Gotfryd(✉), Marek Klonowski, and Dominik Pająk

Department of Computer Science, Wrocław University of Science and Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
{karol.gotfryd,marek.klonowski,dominik.pajak}@pwr.edu.pl

**Abstract.** We consider the following problem – a group of mobile agents perform some task on a terrain modeled as a graph. In a given moment of time an adversary gets access to the graph and agents' positions. Shortly before adversary's observation the devices have a chance to relocate themselves in order to hide their initial configuration, as the initial configuration may possibly reveal to the adversary some information about the task they performed. Clearly agents have to change their locations in possibly short time using minimal energy. In our paper we introduce a definition of a *well hiding* algorithm in which the starting and final configurations of the agents have small mutual information. Then we discuss the influence of various features of the model on running time of the optimal well hiding algorithm. We show that if the topology of the graph is known to the agents, then the number of steps proportional to the diameter of the graph is sufficient and necessary. In the unknown topology scenario we only consider a single agent case. We first show that the task is impossible in the deterministic case if the agent has no memory. Then we present a polynomial randomized algorithm. Finally in the model with memory we show that the number of steps proportional to the number of edges of the graph is sufficient and necessary. In some sense we investigate how complex is the problem of "losing" information about location (both physical and logical) for different settings.

**Keywords:** Location hiding · Mobile agents · Random walks · Graphs

## 1 Introduction

In the present paper we investigate how to hide our location quickly with minimal effort. As our primary motivation we focus on networks consisting of mobile objects, but we believe that our results can also be applied for efficient "losing" information about current state in various systems, even non-physical.

Let us consider a group of mobile devices or sensors called *agents* performing a task on a given area. The task could be for example collecting/detecting some

The work of the second author was supported by Polish National Science Center grant 2013/09/B/ST6/02258. The work of the third author was supported by Polish National Science Center grant 2015/17/B/ST6/01897.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 174–192, 2017.
https://doi.org/10.1007/978-3-319-72050-0_11

valuable resource, mounting detectors or installing mines. In all aforementioned examples the system's owner may want to hide the location of the agents against an adversary observing the terrain from the satellite or a drone. That is, location of the devices may leak sensitive information to the adversary. If we assume that the adversary's surveillance of the terrain is permanent and precise then clearly no information can be concealed. Hence in our scenario there are periods of time when the adversary cannot observe the system during which the actual tasks are performed. Upon the approaching adversary, the devices launch an algorithm to *hide* their *location*, i.e. change their positions to mislead the observer. Clearly in many real life scenarios the additional movement dedicated for hiding their previous position should be possibly short for the sake of saving energy and time. It is also clear that the devices may want to return to their original positions in order to resume their activities when the adversary stops surveillance. On the other hand it is intuitively clear that a very short move may be not sufficient for "losing" the information about the starting positions.

The outlined description is an intuitive motivation for the research presented in our paper. **Exactly** the same problem can be however considered in many other settings when we demand "quick" reconfiguration of a system such that the observed configuration should say possibly small about the initial state. For that reason we decided to use quite general mathematical model, where the agents are placed in vertices of a graph and can move only through the edges (single edge in a single round). Our aim is to design an algorithm that governs the agents' movement to change their initial locations in such a way that the adversary given the final assignment of agents cannot learn their initial positions.

At hand one can point the following strategy – every agent chooses independently at random some vertex on the graph and moves to this location. Clearly (but informally) the new location does not reveal any information about the initial one and the initial locations of agents are perfectly hidden from the adversary. The same effect can also be obtained if all agents go to a single, fixed in advance vertex. In this case again the final and initial configurations are stochastically independent. These strategies require however that agents know the topology of the graph. Intuitively, similar effect can be achieved if each agent starts a random walk and stops in a vertex after some number of steps. In this approach, the knowledge of the graph is **not** necessary, however one can see that the state after any number of steps reveals some knowledge about the initial positions (at least in some graphs). Moreover this strategy requires randomization. To summarize, there are many different methods for hiding the initial locations. It turns out that possible solutions and their efficiency depend greatly on the assumed model – if the graph is known to the agents, what memory is available, if the agents can communicate and if they have access to a source of random bits. Our paper formalizes this problem and discusses its variants in chosen settings.

**Organization of the Paper.** In Sect. 2 we describe the problem and the formal model. Section 3 summarizes the obtained results. The most important related work is mentioned in Sect. 4. In Sect. 5 we present results for the model wherein stations know the topology of the graph representing the terrain. We show both

optimal algorithms as well as respective lower bounds. The case with unknown topology is discussed in Sect. 6. We conclude in Sect. 7. Some basic facts and definitions from Information Theory and theory of Markov chains are recalled in Appendix 1 and Appendix 2, respectively.

## 2 Model

**The Agents in the Network.** We model the network as a simple, undirected, connected graph with $n$ vertices, $m$ edges and diameter $D$. The nodes of the graph are uniquely labeled with numbers $\{1, 2, \ldots, n\}$. We have also $k \geq 1$ agents representing mobile devices. Time is divided into synchronous rounds. At the beginning of each round each agent is located in a single vertex. In each round the agent can change its position to any of neighboring vertices. We allow many agents to be in a single vertex in the same round. The agents need to locally distinguish the edges in order to navigate in the graph hence we assume that the edges outgoing from a node with degree $d$ are uniquely labeled with numbers $\{1, 2, \ldots, d\}$. We assume no correlation between the labels on two endpoints of any node. A graph with such a labeling is sometimes called *port-labeled*.

When an agent is located in a vertex we assume that it has access to the degree of the node and possibly the value or the estimate of $n$ and to some internal memory sufficient for local computations it performs. In our paper we consider various models of mobile agents depending on the resources at their disposal. This will involve settings where the devices have or have not an access to a source of random bits and they are given a priori the topology of the network or they have no such knowledge. In the latter case we will consider two different scenarios depending on whether the agent has an access to operational memory that remains intact when it traverses an edge or its memory is very limited and does not allow to store any information about the network gathered while it moves from one vertex to another.

Our primary motivation is the problem of physical hiding of mobile devices performing tasks in some terrain. Nevertheless, our work aims for formalizing the problem of losing information on agents' initial placement in a given network. Thus, we focus on proposing a theoretical model related to the logical topology.

**Model of the Adversary.** From the adversary's point of view, the agents are indistinguishable and the nodes of the underlying graph are labeled. The assumption on indistinguishability is adequate for systems with very similar devices. Thus the state of the system in a given round $t$ can be seen as a graph $G$ and a function $n_t(v)$ denoting the number of agents located at node $v$. Let $X_t$, $t \in \{0, 1, \ldots\}$, represents the state of the network at the beginning of $t^{\text{th}}$ round.

We assume that in round 0 the agents complete (or interrupt due to approaching adversary) their actual tasks and run *hiding algorithm* $\mathcal{A}$ that takes $T$ rounds. Just after the round $T$ the adversary is given the final state $X_T$ and, roughly speaking, its aim is to learn as much as possible about the initial state $X_0$. That is, the adversary gets an access to a single configuration (representing a single

view of the system). Note that the adversary may have some *a priori* knowledge that is modeled as a distribution of $X_0$. In randomized settings the adversary has no information about agents' local random number generators. On the other hand, the aim of agents is to make learning the adversary $X_0$ from $X_T$ impossible for **any** initial state (or distributions of states).[1] Moreover the number of rounds $T$ should be as small as possible (we need to hide the location quickly). We also consider *energy complexity* understood as the maximal number of moves (i.e. moving to a neighboring vertex) over all agents in the execution of $\mathcal{A}$. Such definition follows from the fact that we need to have all agents working and consequently we need to protect the most loaded agent against running out of batteries. As we shall see, in all cases considered in this paper the energy complexity is very closely related to the time of getting to the "safe" configuration by all devices, namely it is asymptotically equal $T$.

**Security Measures.** Let $X_0$ be a random variable representing the knowledge of the adversary about the initial state and let $X_T$ denotes the final configuration of the devices after executing algorithm $\mathcal{A}$. We aim to define a measure of efficiency of algorithm $\mathcal{A}$ in terms of location hiding. In case of problems based on "losing" knowledge, there is no single, commonly accepted definition. This is reflected in dozens of papers including [6] and [18]. Nevertheless the good security measure needs to estimate "how much information" about $X_0$ is in $X_T$.

Let $X \sim \mathcal{L}$ be a random variable with probability distribution $\mathcal{L}$. We denote by $E[X]$ the expected value of $X$. By $\mathrm{Unif}(A)$ we denote the uniform distribution over the set $A$ and by $\mathrm{Geo}(p)$ the geometric distribution with parameter $p$. An event $E$ occurs with high probability (w.h.p.) if for an arbitrary constant $\alpha > 0$ we have $\Pr[E] \geq 1 - \mathrm{O}(n^{-\alpha})$. Let $H(X)$ denotes the entropy of the random variable $X$, $H(X|Y)$ denotes conditional entropy and $I(X,Y)$ mutual information. All that notations and definitions are recalled in Appendix 1.

Our definition is based on the following notion of *normalized mutual information*, also known as *uncertainity coefficient* (see e.g. [20], Chap. 14.7.4)

$$U(X|Y) = \frac{I(X,Y)}{H(X)}.$$

From the definition of mutual information it follows that $U(X|Y) = 1 - \frac{H(X|Y)}{H(X)}$ and $0 \leq U(X|Y) \leq 1$. The uncertainity coefficient $U(X|Y)$ takes the value 0 if there is no association between $X$ and $Y$ and the value 1 if $Y$ contains all information about $X$. Intuitively, it tells us what portion of information about $X$ is revealed by the knowledge of $Y$. $H(X) = 0$ implies $H(X|Y) = 0$ and we may use the convention that $U(X|Y) = 0$ in that case. Indeed, in such case we have stochastic independence between $X$ and $Y$ and the interpretation in terms of information hiding can be based on the simple observation that $H(X) = 0$ means that there is nothing to reveal about $X$ (as we have full knowledge of $X$) and $Y$ does not give any extra information.

[1] That is, we consider the worst case scenario implying strongest security guarantees.

**Definition 1.** *The algorithm $\mathcal{A}$ is $\varepsilon$-hiding if for any distribution of the initial configuration $X_0$ with non-zero entropy (i.e. $H(X_0) > 0$) and for* ***any*** *graph $G$ representing the underlying network*

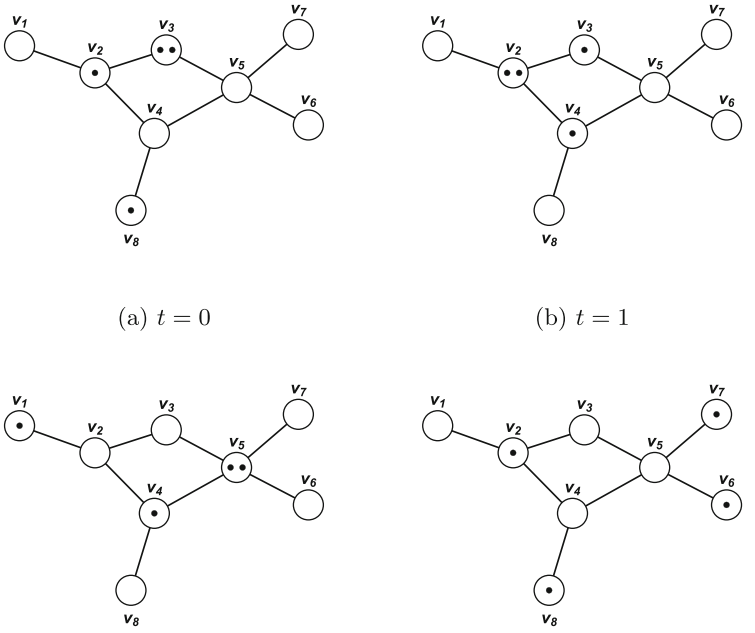
$$U(X_0|X_T) = \frac{I(X_0, X_T)}{H(X_0)} \leq \varepsilon, \tag{1}$$

*where $X_T$ is the state just after the execution of the algorithm $\mathcal{A}$.*

**Definition 2.** *We say that the algorithm $\mathcal{A}$ is*

- *well hiding if it is $\varepsilon$-hiding for some $\varepsilon(n) = \varepsilon \in \mathrm{o}(1)$;*
- *perfectly hiding if it is 0-hiding.*

Intuitively, this definition says that the algorithm works well if the knowledge of the final state reveals only a very small fraction of the total information about the initial configuration regardless of the distribution of initial placement of devices. Let us mention that these definitions state that any hiding algorithm should work well regardless of the network topology. If an algorithm $\mathcal{A}$ is $\varepsilon$-hiding, then for **any** simple connected graph $G$ and for **any** probability distribution of agents' initial positions $X_0$ with non-zero entropy the final configuration $X_T$




(a) $t = 0$ (b) $t = 1$

(c) $t = 2$ (d) $t = 3 = T$

**Fig. 1.** Positions of $k = 4$ agents (represented by black dots) in consecutive steps of sample execution of location hiding algorithm in a network with $n = 8$ nodes.

after $\mathcal{A}$ terminates should fulfill (1). Notice also that there are some cases when it is not feasible to hide the initial location in a given graph. Assuming the adversary knows the agents' initial distribution $X_0$, $H(X_0) = 0$ means that the agents with probability 1 are initially placed in some fixed locations which are known to the adversary. In particular, this is the case when the graph has only one vertex (it can be a model of system with exactly one state). All devices must be then located in that vertex and no hiding algorithm exists for this setting.

The main idea of location hiding algorithms is depicted in Fig. 1. The agents are initially placed in vertices of a graph $G$ (Fig. 1a) according to some known distribution of initial state $X_0$. In each step every agent located in some vertex $v_i$ can move along an edge incident to $v_i$ or stay in $v_i$. After $T$ steps the algorithm terminates resulting in a final configuration $X_T$ (Fig. 1d). Our goal is to ensure that any adversary observing the positions $X_T$ of devices after execution of an location hiding algorithm can infer as small information as possible about their actual initial placement $X_0$, regardless of $G$ and the distribution of $X_0$.

## 3 Our Results

Most of our results apply to the single-agent case. We first show that if the topology is known then any well-hiding algorithm in a graph with $n$ nodes, $m$ edges and diameter $D$ requires $\Omega(D)$ steps and there exists a perfectly hiding algorithm that needs $O(D)$ steps. Then we generalize this result to multi-agent scenario. Secondly we consider the case with unknown network topology. We show that in the model with no memory there exists no deterministic well hiding algorithm and for the randomized setting we present a well-hiding algorithm whose expected running time is $\widetilde{O}(n^3)$ w.h.p. Finally if the agents have unlimited memory then $\Theta(m)$ steps is sufficient and necessary for well-hiding algorithms. Table 1 summarizes our results.

**Table 1.** Overview of our results

| | | Deterministic | Randomized |
|---|---|---|---|
| Known topology | | $\Theta(D)$ (Theorem 1) | $\Theta(D)$ (Theorem 1) |
| Unknown topology | No memory | impossible (Theorem 3) | $\widetilde{O}(n^3)$ w.h.p. (Theorem 2) |
| | Unlimited memory | $\Theta(m)$ (Theorems 4 and 5) | $\Theta(m)$ (Theorems 4 and 5) |

Let us mention that in the considered models it is feasible to "lose" information about the initial state not only in a randomized manner, but also fully deterministically. As we shall show, the algorithms are completely different. We find this property somehow surprising. Moreover, let us note that possible rate of losing information as well as the adequate algorithms strongly depend on the assumed model and agents' capabilities (knowledge of the topology, memory).

## 4 Previous and Related Work

The problems of security and privacy protection in distributed systems have received a lot of attention. Various security aspects of such systems have been extensively discussed and a lot of novel solutions for some practical settings have been proposed over the last years. One of the major examples is the problem of designing routing protocols for wireless ad hoc networks which can hide the network topology from potential external and internal adversaries (see e.g. [16, 24]). The goal of such protocols is to find reliable routes between the source and destination nodes in the network which are as short as possible, reducing exposure of the network topology to malicious nodes by data transmitted in the packets. This will prevent adversaries (at least to some extent) for launching some kinds of attacks requiring the knowledge of the network topology which may be particularly harmful for the whole network and the tasks performed.

Another important line of research is assuring privacy of the users of mobile applications relying on location-based services and hence gathering information of their location. The examples are applications providing various information related to the user's current location (e.g. real-time traffic information, places to visit) or activity-based social networks where users share the information about location-based activities they perform (cf. [19]). Various protocols for protecting location data together with some formal models and privacy metrics were proposed (see e.g. [12,14]). However, in some cases the performance of designed protocols is evaluated only experimentally and the discussion of their security properties is informal, without referring to any theoretical model (cf. [16,19]).

To the best of our knowledge, there is no rigid and formal analysis on the problem of location hiding in graphs and it has never been studied before in the context considered in this paper. The problems of ensuring security and privacy in distributed systems mentioned above are similar to our only to a certain extent. The aim of our approach is to propose a general formal model of hiding the positions of a set of mobile agents from an external observer and consider its basic properties and limitations. However, the problem considered by us is closely related to some of the most fundamental agent-based graph problems.

First of all observe the relation to the exploration that comes from the global nature of our problem. Clearly if the agent has at least logarithmic memory then we can use algorithms for graph exploration. Indeed, since the graph is labeled, it is sufficient to explore the graph and move to a vertex with minimum ID. Hence the vast body of literature about exploration in various models applies to our problem. In particular there exist polynomial deterministic algorithms (using Universal Sequences) that need only logarithmic memory [1,21].

In the randomized setting, location hiding becomes related to the problem of reaching the stationary distribution by a Markov Chain (Mixing Time) as well as visiting all the states (Cover Time), i.e. the expected number of steps that are needed by a random walk to visit all the vertices. It is known that for a (unbiased) random walk, the cover time is between $\Omega(n \log n)$ [10] and $O(n^3)$ [11], depending on the underlying graph structure. There exist biased random walks that achieve worst-case cover time of $\widetilde{O}(n^2)$ [17], however in order to implement

them the agent requires an access to some memory to acquire information necessary to compute the transition probabilities. It has been recently shown that in some graphs multiple random walks are faster than a single one [2,8,9]. Another interesting line of work is deriving biased random walks [3,23].

## 5 Location Hiding for Known Topology

Let us first focus on the setting where the topology of the underlying network is known to the agents and consider one of the simplest possible protocols, namely every mobile agent goes from its initial positions to some fixed vertex $v^* \in V$ (this is possible, because in the considered scenario the vertices in the graph have unique labels known to all the agents). One can easily see that this simple protocol is perfectly hiding. Indeed, regardless of the distribution of the agents' initial placement, after executing the protocol all devices are always located in the same vertex known in advance. Hence, $X_T$ and $X_0$ are independent and $I(X_0, X_T) = 0$ (and therefore $U(X_0|X_T) = 0$, as required). But this approach leads to the worst case time and energy complexity for a single device of order $\Theta(D)$, where $D$ is the graph diameter. Appropriate selection of the vertex $v^*$ as an element of the graph center can reduce the worst case complexity only by a constant, but it does not change its order. The natural question that arises in this context is whether there exist a perfectly hiding (or at least well hiding) protocol that requires asymptotically smaller number of rounds for ensuring privacy than the simple deterministic protocol discussed above. In general, we are interested in determining the minimal number of steps required by any location hiding protocol in considered scenarios for ensuring a given level of security (in terms of the amount of information being revealed) for arbitrary distribution of initial configuration of the agents and for arbitrary underlying network.

### 5.1 Single Agent Scenario

Let us consider the simple scenario where there is only one mobile device in the network located in some vertex $v \in V$ according to some known probability distribution $\mathcal{L}$ over the set of vertices. Assume that the network topology is known to the agent. Our goal is to find the lower bound on the number of steps that each well hiding protocol requires to hide the original location of the device in this scenario for arbitrary graph $G$ and initial distribution $\mathcal{L}$.

We will start with a general lemma showing that if within $t$ steps the sets of vertices visited by the algorithm starting from two different vertices are disjoint with significant probability, then the algorithm is not well hiding within time $t$.

**Lemma 1.** *Let $\mathcal{A}$ be any hiding algorithm and $G = (V, E)$ be an arbitrary graph. Suppose that for some $t > 0$ and some positive constant $\gamma$ there exist two distinct vertices $u, v \in V$ s.t. with probability at least $1/2+\gamma$ the following property holds: sets $V_1$ and $V_2$ of vertices reachable after execution of $t$ steps of $\mathcal{A}$ when starting from $u$ and $v$, respectively, are disjoint. Then $\mathcal{A}$ is not well hiding in time $t$.*

*Proof.* Fix an arbitrary graph $G = (V, E)$ with $|V| = n$ and hiding algorithm $\mathcal{A}$. Let $u, v \in V$ be two vertices such that the sets $V_1$ and $V_2$ of possible location of the agent after performing $t$ steps of $\mathcal{A}$ when starting in $u$ and $v$, respectively, are disjoint with probability at least $1/2+\gamma$ for some constant $\gamma > 0$ regardless of the starting point, i.e. $\Pr[\xi_V = 1] \geq 1/2+\gamma$, where $\xi_V$ is an indicator random variable of the event $V_1 \cap V_2 = \emptyset$. Consider the following two-point distribution $\mathcal{L}$ of the agents' initial location $X_0$: $\Pr[X_0 = u] = \Pr[X_0 = v] = 1/2$, $\Pr[X_0 = w] = 0$ for $w \in V \setminus \{u, v\}$. We will prove that such $\mathcal{A}$ does not ensure that the initial position $X_0$ of the device is well hidden at time $t$ when $X_0 \sim \mathcal{L}$.

Because $H(X_0) = 1$, $U(X_0|X_t) = I(X_0, X_t)$ and it suffices to show that the mutual information $I(X_0, X_t) \geq \eta > 0$ for some positive constant $\eta$. This is equivalent to $H(X_0|X_t) \leq 1 - \eta$, as follows from Fact 4. Clearly, for $y \in V_1$

$$\Pr[X_0 = u|X_t = y] \geq \Pr[X_0 = u, \xi_V = 1|X_t = y] \geq 1/2 + \gamma \tag{2}$$

and the same holds after replacing $u$ with $v$. Moreover $\Pr[X_0 = v|X_t = y] = 1 - \Pr[X_0 = u|X_t = y]$. Denoting $\Pr[X_0 = u|X_t = y]$ by $p_{u|y}$ we have

$$\begin{aligned} H(X_0|X_t) &= -\sum_{y \in V} \Pr[X_t = y] \sum_{x \in V} \Pr[X_0 = x|X_t = y] \log(\Pr[X_0 = x|X_t = y]) \\ &= -\sum_{y \in V} \Pr[X_t = y] \left(p_{u|y} \log(p_{u|y}) + (1 - p_{u|y}) \log(1 - p_{u|y})\right). \end{aligned} \tag{3}$$

Let us consider the function $f(p) = -(p \log(p) + (1-p) \log(1-p))$ for $p \in (0, 1)$. Clearly, $\lim_{p \to 0} f(p) = \lim_{p \to 1} f(p) = 0$ and $f(p)$ has its unique maximum on the interval (0,1) equal to 1 at $p = 1/2$. From (2) we have $(\forall y \in V_1)(p_{u|y} \geq 1/2 + \gamma)$ and $(\forall y \in V_2)(p_{u|y} \leq 1/2 - \gamma)$. Therefore, there exists some positive constant $\eta$ such that $f(p_{u|y}) \leq 1 - \eta$. From the definition of the sets $V_1$ and $V_2$ we also have $\Pr[X_t = y \notin V_1 \cup V_2] = 0$. Using these facts, (3) can be rewritten as

$$\begin{aligned} H(X_0|X_t) &= \sum_{y \in V_1} \Pr[X_t = y] f(p_{u|y}) + \sum_{y \in V_2} \Pr[X_t = y] f(p_{u|y}) \\ &\leq (1 - \eta) \Pr[X_t \in V_1 \cup V_2] = 1 - \eta \end{aligned}$$

for some constant $\eta > 0$, as required. Hence, the lemma is proved. □

From the Lemma 1 we get the lower bound of $\Omega(D)$ on the expected number of steps needed by any well hiding algorithm in the model with known topology. Note that the lower bound matches the simple $O(D)$ upper bound.

**Theorem 1.** *For a single agent and known network topology and for an arbitrary graph $G$ there exists a distribution $\mathcal{L}$ of agent's initial position such that any well hiding algorithm $\mathcal{A}$ needs to perform at least $\lfloor D/2 \rfloor$ steps with probability $c \geq 1/2 - o(1)$, where $D$ is the diameter of $G$.*

*Proof.* We will show that for each graph $G$ there exist a distribution of the initial state of the mobile agent such that each well hiding algorithm $\mathcal{A}$ needs at least $\lfloor D/2 \rfloor$ rounds with some probability $c \geq 1/2 - \mathrm{o}(1)$.

Fix an arbitrary graph $G = (V, E)$ with $|V| = n$. Let $u, v \in V$ be two vertices such that $d(u, v) = D$. Denote by $\delta = \lfloor D/2 \rfloor$ and consider the following two-point distribution $\mathcal{L}$ of the agents' initial location $X_0$: $\Pr[X_0 = u] = \Pr[X_0 = v] = 1/2$, $\Pr[X_0 = w] = 0$ for $w \in V \setminus \{u, v\}$. Suppose that some hiding algorithm $\mathcal{A}$ terminates with probability at least $1/2 + \gamma$ for some constant $\gamma > 0$ after $T < \delta$ steps regardless of the starting point, i.e. $\Pr[T < \delta] \geq 1/2 + \gamma$.

Obviously there is no $z \in V$ such that $d(u, z) < \delta$ and $d(v, z) < \delta$ (if so, $D = d(u, v) < 2\lfloor D/2 \rfloor \leq D$ and we will get a contradiction). Let us define $B(u, \delta) = \{y \in V \colon d(u, y) < \delta\}$ and $B(v, \delta) = \{y \in V \colon d(v, y) < \delta\}$. It is clear that $B(u, \delta) \cap B(v, \delta) = \emptyset$. From the assumptions on the running time of $\mathcal{A}$ with probability at least $1/2 + \gamma$ the sets $V_1$ and $V_2$ of vertices reachable from $u$ and $v$, respectively, fulfills $V_1 \subseteq B(u, \delta)$ and $V_2 \subseteq B(v, \delta)$, therefore they are disjoint. Hence it suffices to apply the results from Lemma 1 to complete the proof. □

### 5.2 Location Hiding for $k$ Agents and Known Network Topology

Let us recall that the energy complexity of an algorithm $\mathcal{A}$ in the multi-agent setting is defined as the **maximal** distance covered (i.e. number of moves) in the execution of $\mathcal{A}$ over all agents. This allows us for direct translation of results from single-device setting, as presented below.

In the general scenario considered in this section a similar result holds as for the single-agent case. Namely, each algorithm which ensures the well hiding property regardless of the distribution of agents' initial placement requires in the worst case $\Omega(D)$ rounds.

**Lemma 2.** *For known network topology and $k > 1$ indistinguishable agents initially placed according to some arbitrary distribution $\mathcal{L}$, any well hiding algorithm for an arbitrary underlying graph $G$ has energy complexity at least $\lfloor D/2 \rfloor$ with probability $c \geq 1/2 - \mathrm{o}(1)$, where $D$ is the diameter of $G$.*

The proof of the Lemma 2 proceeds in the same vein as in Theorem 1. We choose two vertices $u, v$ in distance $D$ and put all agents with probability $1/2$ in any of these vertices. Denoting by $T_i$, $1 \leq i \leq k$, the number of steps performed by the agent $i$ and by $T = \max_{1 \leq i \leq k} T_i$ the energy complexity of the algorithm we consider a hiding algorithm $\mathcal{A}$ such that $\Pr[T < \delta] \geq 1/2 + \gamma$ for $\delta = \lfloor D/2 \rfloor$ and some positive constant $\gamma > 0$. The only difference is that instead of the sets $B(u, \delta) = \{y \in V \colon d(u, y) < \delta\}$ and $B(v, \delta) = \{y \in V \colon d(v, y) < \delta\}$ itself we consider the subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ of the state space consisting of such states that all of the agents are located only in the vertices from the set $B(u, \delta)$ or $B(v, \delta)$, respectively. Similar calculations as previously led to the conclusion that any such algorithm cannot ensure the well hiding property.

## 6 Location Hiding for Unknown Topology

### 6.1 No Memory

If no memory and no information about the topology is available, but the agent is given access to a source of randomness, it can perform a random walk in order to conceal the information about its starting position. However, the agent would not know when to stop the walk. If in each step it would choose to terminate with probability depending on the degree of the current node, one could easily construct an example in which the agent would not move far from its original position (with respect to the network size). Hence in this section we assume that the size of the network is known. Then the problem becomes feasible. Consider the following Algorithm 1: in each step we terminate with probability $q$ (roughly $n^{-3}$) and with probability $1-q$ we make one step of a lazy random walk. We will choose $q$ later. Let us note that letting the walk to stay in current vertex with some fixed constant probability is important for ensuring aperiodicity of the Markov chain (see e.g. [13,15]). Otherwise we can easily provide an example where such algorithm does not guarantee the initial position to be hidden. Namely, consider any bipartite graph and any initial distribution s.t. the agent starts with some constant probability either in a fixed *black* or *white* vertex. If the adversary is aware only of the running time (i.e. the number of steps the agent performed), when observing the network after $T$ steps it can with probability 1 identify agent's initial position depending on $T$ is even or odd. Nevertheless, the probability of remaining in a given vertex can be set to arbitrary constant $0 < c < 1$. For the purposes of analysis we let $c = 1/2$ which leads to the classical definition of lazy random walk (see Definition 10 and Fact 8 in Appendix 2).

**Algorithm 1.** $\mathcal{A}(q)$ [randomization, no memory, no topology, knowledge of $n$]

**In each round:**

1: With probability $q$: terminate the algorithm.
2: With probability $\frac{1}{2}$: remain in the current vertex until the next round.
3: With probability $\frac{1}{2} - q$: move to a neighbor chosen uniformly at random.

**Theorem 2.** *The algorithm $\mathcal{A}(q)$ based on random walk with termination probability $q = \frac{f(n)}{n^3 \log h(n)}$ for any fixed $f(n) = \mathrm{o}(1)$ and $h(n) = \omega\left(\max\{n^2, \frac{1}{H(X_0)}\}\right)$ is well hiding for any graph $G$ and any distribution of agent's initial location $X_0$.*

*Proof.* Fix $\varepsilon > 0$. Let $t_{\mathrm{mix}}(\varepsilon)$ denote the mixing time and $\pi$ the stationary distribution of the random walk performed by the algorithm according to Definition 9. We will choose the exact value for $\varepsilon$ later. Let $X_0$ and $X_T$ be the initial and final configuration, respectively. To prove the lemma it suffices to show that $\frac{H(X_0|X_T)}{H(X_0)} = 1 - \mathrm{o}(1)$, what is equivalent to $\lim_{n\to\infty} \frac{H(X_0|X_T)}{H(X_0)} = 1$. This implies that $U(X_0|X_T) = \mathrm{o}(1)$ as required by Definition 2. Let $\xi_\varepsilon = \mathbf{1}[T > t_{\mathrm{mix}}(\varepsilon)]$ be

the indicator random variable taking value 1 if $T > t_{\mathrm{mix}}(\varepsilon)$ and 0 otherwise. We need to ensure that the algorithm $\mathcal{A}$ will stop with probability at least $1 - \mathrm{o}(1)$ after $t_{\mathrm{mix}}(\varepsilon)$ steps. The time $T$ when $\mathcal{A}$ terminates follows Geo($q$) distribution, hence $\Pr[\xi_\varepsilon = 1] = (1-q)^{t_{\mathrm{mix}}(\varepsilon)}$. Letting $q = f(n)/t_{\mathrm{mix}}(\varepsilon)$ for some $f(n) = \mathrm{o}(1)$ implies $\Pr[\xi_\varepsilon = 1] = 1 - \mathrm{o}(1)$, as required.

Let us consider $H(X_0|X_T)$. By Fact 1 (see Appendix 1) we have

$$\begin{aligned} H(X_0|X_T) &\geq H(X_0|X_T, \xi_\varepsilon) \geq H(X_0|X_T, \xi_\varepsilon = 1) \Pr[\xi_\varepsilon = 1] \\ &= (1 - \mathrm{o}(1))\, H(X_0|X_T, \xi_\varepsilon = 1) \geq (1 - \mathrm{o}(1))\, H\left(X_0|X_{t_{\mathrm{mix}}(\varepsilon)}\right), \end{aligned}$$

where the last inequality follows directly from Fact 9 in Appendix 1.

Let $p_0(x) = \Pr[X_0 = x]$, $p_t(y) = \Pr[X_{t_{\mathrm{mix}}(\varepsilon)} = y]$, $p_0(x|y) = \Pr[X_0 = x|X_{t_{\mathrm{mix}}(\varepsilon)} = y]$ and $p_t(y|x) = \Pr[X_{t_{\mathrm{mix}}(\varepsilon)} = y|X_0 = x]$. As $p_0(x|y) = \frac{p_t(y|x)}{p_t(y)} p_0(x)$,

$$\begin{aligned} H\left(X_0|X_{t_{\mathrm{mix}}(\varepsilon)}\right) &= -\sum_{y\in V} p_t(y) \sum_{x\in V} p_0(x|y) \log p_0(x|y) \\ &= -\sum_{y\in V}\sum_{x\in V} p_t(y|x) p_0(x) \log p_0(x) \\ &\quad -\sum_{y\in V}\sum_{x\in V} p_t(y|x) p_0(x) \log \frac{p_t(y|x)}{p_t(y)}. \end{aligned} \tag{4}$$

The properties of mixing time imply that there exist $\{\varepsilon_y^{(1)}\}_{y\in V}$ and $\{\varepsilon_y^{(2)}\}_{y\in V}$ such that $\sum_{y\in V} \varepsilon_y^{(i)} \leq 2\varepsilon$ for $i \in \{1, 2\}$ and $\pi(y) - \varepsilon_y^{(1)} \leq p_t(y|x) \leq \pi(y) + \varepsilon_y^{(1)}$ and $\pi(y) - \varepsilon_y^{(2)} \leq p_t(y) \leq \pi(y) + \varepsilon_y^{(2)}$. Let $\varepsilon_y = \max\{\varepsilon_y^{(1)}, \varepsilon_y^{(2)}\}$. As for any $y \in V$ $\pi(y) \geq 1/n^2$, letting $\varepsilon$ being arbitrary $\varepsilon(n) = \mathrm{o}\left(\min\{\frac{1}{n^2}, H(X_0)\}\right)$ we get

$$\frac{p_t(y|x)}{p_t(y)} \leq \frac{\pi(y) + \varepsilon_y}{\pi(y) - \varepsilon_y} = 1 + \mathrm{o}\left(\min\{1, H(X_0)\}\right).$$

Thus, the above relations allow us to find the lower bound on the conditional entropy $H\left(X_0|X_{t_{\mathrm{mix}}(\varepsilon)}\right)$. The first sum in (4) gives us

$$\begin{aligned} -\sum_{y\in V}\sum_{x\in V} p_t(y|x) p_0(x) \log p_0(x) &\geq \sum_{y\in V} (\pi(y) - \varepsilon_y) H(X_0) \geq H(X_0)(1 - 4\varepsilon) \\ &= H(X_0)(1 - \mathrm{o}(1)), \end{aligned} \tag{5}$$

whereas the second sum can be expressed as

$$\begin{aligned} -\sum_{y\in V}\sum_{x\in V} p_t(y|x) p_0(x) \log \frac{p_t(y|x)}{p_t(y)} &= -\sum_{x\in V} p_0(x) \sum_{y\in V} p_t(y|x) \log \frac{p_t(y|x)}{p_t(y)} \\ &= -\sum_{x\in V} p_0(x) D\left(p_t(y|x)||p_t(y)\right), \end{aligned}$$

where $D(\cdot||\cdot)$ is relative entropy recalled in Definition 5 in Appendix 1.

Applying the upper bound on the relative entropy from Fact 3 we get

$$\begin{aligned}\sum_{x\in V} p_0(x) D\left(p_t(y|x)||p_t(y)\right) &\leq \sum_{x\in V} p_0(x)\frac{1}{\ln 2}\left(\sum_{y\in V}\frac{(p_t(y|x))^2}{p_t(y)} - 1\right)\\ &\leq \frac{1}{\ln 2}\sum_{x\in V} p_0(x)\sum_{y\in V}\left(\frac{(\pi(y)+\varepsilon_y)^2}{\pi(y)-\varepsilon y} - \pi(y)\right) \qquad (6)\\ &= \mathrm{o}\left(H\left(X_0\right)\right).\end{aligned}$$

Combining the estimations (5) and (6) we obtain $H\left(X_0|X_{t_{\mathrm{mix}}(\varepsilon)}\right) \geq H\left(X_0\right)(1-\mathrm{o}\left(1\right)) - \mathrm{o}\left(H\left(X_0\right)\right)$, what results in

$$\frac{H\left(X_0|X_T\right)}{H\left(X_0\right)} \geq \frac{H\left(X_0\right)\left(1-\mathrm{o}\left(1\right)\right) - \mathrm{o}\left(H\left(X_0\right)\right)}{H\left(X_0\right)} = 1 - \mathrm{o}\left(1\right),$$

as required.

In the above we have set $q = f(n)/t_{\mathrm{mix}}\left(\varepsilon\right)$ for arbitrary fixed $f(n) = \mathrm{o}\left(1\right)$ and $\varepsilon = \mathrm{o}\left(\min\{n^{-2}, H\left(X_0\right)\}\right)$. From Fact 7 and Fact 8 we have $t_{\mathrm{mix}}\left(\varepsilon\right) \leq n^3 \log \varepsilon^{-1}$. Hence, there exists some $g(n) = \omega\left(\max\{n^2, H\left(X_0\right)^{-1}\}\right)$ dependent on $\varepsilon$ such that $t_{\mathrm{mix}}\left(\varepsilon\right) \leq n^3 \log g(n)$ and $q = \left(h(n)\cdot n^3 \log g(n)\right)^{-1}$, where $h(n) = 1/f(n) = \omega\left(1\right)$. □

As previously mentioned, the running time $T$ of the considered hiding algorithm follows geometric distribution with parameter $q$, hence the expected running time is $E\left[T\right] = 1/q = h(n)\cdot n^3 \log g(n)$, where $h(n)$ and $g(n)$ are as in the proof of Theorem 2. If $H(X_0) = \Omega\left(\frac{1}{n^2}\right)$, as in the case of most distribution considered in practice, we can simply select $f(n)$ to be some function decreasing to 0 arbitrary slowly and $\varepsilon$ such that $g(n) = cn^3 \log n$ for some constant $c > 0$. In such cases the entropy of the distribution of agent's initial position has no impact on the upper bound on the algorithm's running time.

Algorithm 1 works also for the scenario with many agents (each agent can run independent walk). The interesting question is whether it is possible to hide the initial state in multi-agent case faster by taking advantage of performing simultaneously many random walks. As the speedup of multiple random walks in any graph remains a conjecture [2], we leave this as an open question.

We conclude this section with a simple observation that the agent must have access to either memory, source of randomness or the topology of the network in order to hide.

**Theorem 3.** *In the model with unknown topology and with no memory there exists no well-hiding deterministic algorithm.*

*Proof.* Take any hiding algorithm. If this algorithm never makes any move it obviously is not well-hiding. Otherwise observe that in the model without memory the move is decided only based on local observations (degree of the node) and some global information (value of $n$), hence every time the agent visits a

node it will make the same decision. Assume that the agent decides to move from a node of degree $d$ via port $p$. We construct a graph from two stars with degree $d$ joined by an edge $e$ with port $p$ on both endpoints. Since the agent has no memory and no randomness it will end up in an infinite loop traversing edge $e$. Hence this algorithm cannot be regarded as well-hiding since it never terminates. □

### 6.2 Unlimited Memory

In this section we assume that the agent is endowed with unlimited memory that remains intact when the agent traverses an edge. We first observe that a standard search algorithm (e.g. DFS) can be carried out in such a model.

**Theorem 4.** *There exists a perfectly hiding algorithm in the model with unlimited memory that needs $O(m)$ steps in any graph.*

*Proof.* The algorithm works as follows: it runs a DFS search of the graph (it is possible if the agent has memory) and moves to the node with minimum ID. □

Now we would like to show that $\Omega(m)$ steps are necessary for any well-hiding algorithm in this model. We will construct a family of graphs such that for any well-hiding algorithm and any $n$ and $m$ we can find a graph with $n$ nodes and $m$ edges in this family such that this algorithm will need on average $\Omega(m)$ steps.

**Theorem 5.** *For a single agent and unknown network topology, for any $n$ and $m$ and any well hiding algorithm $\mathcal{A}$ there exists a port labeled graph $G$ with $n$ vertices and $m$ edges representing the network and a distribution $\mathcal{L}$ of agent's initial position on which the agent needs to perform $\Omega(m)$ steps in expectation.*

*Proof.* If $m = O(n)$ we can construct a graph in which $D = \Omega(m)$ and use Theorem 1. Now assume that $m = \omega(n)$ and consider a graph constructed by connecting a chain of $y$ cliques of size $x$. If $m = \omega(n)$ we can find such $x, y$ that $x = \Theta(m/n)$ and $y = \Theta(n^2/m)$. The adjacent cliques are connected by adding an vertex on two edges (one from each clique) and connecting these new vertices by an additional edge. We call this edge by *bridge* and the vertices adjacent to a bridge by *bridgeheads*. Let $\mathcal{G}_{x,y}$ be the family of all such chains of cliques on $n$ nodes and $m$ edges (note that we take only such chains in which an edge has at most one bridgehead). We want to calculate the expected time that $\mathcal{A}$ needs to reach the middle of the chain (if $y$ is even then it is the middle bridge, otherwise it is the middle clique) on a graph chosen uniformly at random from $\mathcal{G}_{x,y}$.

When the agent is traversing edges of the clique, each edge contain a bridgehead with probability $\left(\binom{x}{2} - 1\right)^{-1}$, hence with probability at least $\frac{1}{2}$ the agent needs to traverse $\left(\binom{x}{2} - 1\right)/2$ different edges. As in each clique bridgeheads are chosen independently, we can choose the constants so that by the Chernoff Bound the time to reach the middle of the chain is $\Omega\left(y \cdot x^2\right) = \Omega\left(m\right)$ with probability at least $\frac{3}{4}$ if $G$ is chosen uniformly at random from $\mathcal{G}_{x,y}$. By symmetry, this holds for both endpoints (reaching middle from the first or the $y$-th clique). Hence,

there exists $G^* \in \mathcal{G}_{x,y}$ such that with probability at least $\frac{3}{4}$ the time of $\mathcal{A}$ to reach the middle from both endpoints is at least $c \cdot m$ for some constant $c > 0$. By Lemma 1, $\mathcal{A}$ is not well-hiding on $G^*$ if the number of steps is at most $c \cdot m$. □

## 7 Conclusions and Further Research

We introduced the problem of location hiding, discussed efficient algorithms and lower bounds for some settings. Nevertheless, some questions are left unanswered.

The model considered by us encompasses wide range of scenarios with large variety of possible agents arrangements. Moreover, some natural classes of graphs may provide reasonable approximation in the cases where terrain should be modeled as a connected region on a plane. Some examples of such graphs are those from families of grid-like graphs. They contain edges joining only those pairs of vertices which are close to each other (in the sense that they have small euclidean distance after embedding the graph on a plane). Increasing the number of vertices leads then to more close resemblance to connected continuous regions. It would be, however, an interesting line of research to fully extend our approach to continuous connected terrains and derive analogous results for that model.

Another line of research is the model with dynamic topology that may change during the execution of the protocol. Similarly, we believe that it would be interesting to investigate the model with the weaker adversary that is given only partial knowledge of the graph topology and the actual assignment of agents. On the other hand, one may study more powerful adversaries being able to observe some chosen part of the network for a given period of time. It would be also worth considering how high level of security can be achieved if each agent is able to perform only O (1) steps. Motivated by the fact that the mobile devices are in fact similar objects, we considered the setting where they are indistinguishable. It would be useful to study the case when the adversary can distinguish between different agents. We also plan to deeper understand the relation of location hiding problem with classic, fundamental problems like rendez-vous or patrolling.

We defined the energy complexity as maximal energetic expenditure over all agents. In some cases, however, it would be more adequate to consider total energy used by all stations. Finally, it would also be interesting to construct more efficient protocols for given classes of graphs with some common characteristic (e.g., lines, trees) and algorithms desired for restricted distributions of $X_0$.

**Acknowledgments.** The authors of this paper would like to thank to anonymous reviewers for their valuable comments, suggestions and remarks.

## Appendix 1: Information Theory

We recall some basic definitions and facts from Information Theory that can be found e.g. in [5]. In all cases below log will denote the base-2 logarithm.

**Definition 3 (Entropy).** *For a discrete random variable* $X \colon \mathcal{X} \to \mathbb{R}$ *the entropy of* $X$ *is defined as* $H(X) = -\sum_{x \in \mathcal{X}} \Pr[X = x] \log \Pr[X = x]$.

**Definition 4 (Conditional entropy).** *If $X\colon \mathcal{X} \to \mathbb{R}$ and $Y\colon \mathcal{Y} \to \mathbb{R}$ are two discrete random variables, we define the conditional entropy as*

$$H(X|Y) = -\sum_{y\in\mathcal{Y}} \Pr[Y=y] \sum_{x\in\mathcal{X}} \Pr[X=x|Y=y] \log \Pr[X=x|Y=y].$$

**Fact 1.** *For any random variables $X$ and $Y$ $H(X|Y) \leq H(X)$ and the equality holds if and only if $X$ and $Y$ are independent.*

**Definition 5 (Relative entropy).** *Let $X$ and $Y$ be two discrete random variables defined on the common space $\mathcal{X}$ with pmf $p(x)$ and $q(x)$, respectively. The relative entropy (Kullback-Leibler distance) between $p(x)$ and $q(x)$ is*

$$D(p||q) = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)}.$$

**Fact 2 (Information inequality).** *Let $p(x)$ and $q(x)$ be probability mass functions of two discrete random variables $X, Y\colon \mathcal{X} \to \mathbb{R}$. Then $D(p||q) \geq 0$ with equality if and only if $\forall x \in \mathcal{X}\ p(x) = q(x)$.*

**Fact 3 (Theorem 1 in [7]).** *Let $p(x)$, $q(x) > 0$ be probability mass functions of two discrete random variables $X$ and $Y$, respectively, defined on the space $\mathcal{X}$. Then*

$$D(p||q) \leq \frac{1}{\ln 2} \left( \sum_{x\in\mathcal{X}} \frac{p^2(x)}{q(x)} - 1 \right).$$

**Definition 6 (Mutual information).** *If $X$ and $Y$ are two discrete random variables defined on the spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively, then the mutual information of $X$ and $Y$ is defined as*

$$I(X,Y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} \Pr[X=x, Y=y] \log \left( \frac{\Pr[X=x, Y=y]}{\Pr[X=x] \Pr[Y=y]} \right). \tag{7}$$

**Fact 4.** *For any discrete random variables $X, Y$*

- $0 \leq I(X,Y) \leq \min\{H(X), H(Y)\}$ *and the first equality holds if and only if random variables $X$ and $Y$ are independent,*
- $I(X,Y) = I(Y,X) = H(X) - H(X|Y) = H(Y) - H(Y|X)$.
- $I(X,Y) = D(p(x,y)||p(x)p(y))$ *where $p(x,y)$ denotes the joint distribution, and $p(x)p(y)$ the product distribution of $X$ and $Y$.*

## Appendix 2: Markov Chains

We recall some definitions and facts from the theory of Markov chains. They can be found e.g. in [5,13,15]. Unless otherwise stated, we will consider only time-homogeneous chains, where transition probabilities do not change with time.

**Definition 7 (Total variation distance).** *For probability distributions $\mu$ and $\nu$ on the space $\mathcal{X}$ we define the total variation distance between $\mu$ and $\nu$ as* $d_{\mathrm{TV}}(\mu,\nu) = \max_{A\subseteq\mathcal{X}} |\mu(A) - \nu(A)|$.

**Fact 5.** *Let $\mu$ and $\nu$ be two probability distributions on common space $\mathcal{X}$. Then we have* $d_{\mathrm{TV}}(\mu,\nu) = \frac{1}{2}\sum_{x\in\mathcal{X}} |\mu(x) - \nu(x)|$.

**Definition 8.** *Let $P^t(x_0,\cdot)$ denote the distribution of an ergodic Markov chain on finite space $\mathcal{X}$ in step $t$ when starting in the state $x_0$. Let $\pi$ be the stationary distribution of $M$. We define* $d(t) = \max_{x\in\mathcal{X}} d_{\mathrm{TV}}(P^t(x,\cdot),\pi)$ *and* $\bar{d}(t) = \max_{x,y\in\mathcal{X}} d_{\mathrm{TV}}(P^t(x,\cdot),P^t(y,\cdot))$.

**Fact 6.** *Let $\mathcal{P}$ be the family of all probability distributions on $\mathcal{X}$. Then*

- $d(t) \le \bar{d}(t) \le 2d(t)$,
- $d(t) = \sup_{\mu\in\mathcal{P}} d_{\mathrm{TV}}(\mu P^t,\pi) = \sup_{\mu,\nu\in\mathcal{P}} d_{\mathrm{TV}}(\mu P^t,\nu P^t)$.

**Definition 9 (Mixing time).** *For an ergodic Markov chain $M$ on finite space $\mathcal{X}$ we define the mixing time as* $t_{\mathrm{mix}}(\varepsilon) = \min\{t\colon d(t) \le \varepsilon\}$ *and* $t_{\mathrm{mix}} = t_{\mathrm{mix}}(1/4)$.

**Fact 7.** *For any $\varepsilon > 0$,* $t_{\mathrm{mix}}(\varepsilon) \le \left\lfloor \log \varepsilon^{-1} \right\rfloor t_{\mathrm{mix}}$.

**Definition 10 (Random walk).** *The random walk on a graph $G = (V,E)$ with $n$ nodes and $m$ edges is a Markov chain on $V$ with transition probabilities*

$$p_{ij} = \Pr[X_{t+1} = v_j | X_t = v_i] = \begin{cases} 1/\deg(v_i), & \text{if } \{v_i,v_j\} \in E, \\ 0, & \text{otherwise.} \end{cases}$$

*The lazy random walk is the random walk which, in every time $t$, with probability $1/2$ remains in current vertex or performs one step of a simple random walk.*

The following Fact 8 gives an upper bound on the mixing time for random walks. It follows e.g. from Theorem 10.14 in [13] and the properties of cover time and its relation to mixing time (see [11]).

**Fact 8.** *For a lazy random walk on an arbitrary connected graph $G$ with $n$ vertices* $t_{\mathrm{mix}} = \mathrm{O}\left(n^3\right)$.

**Fact 9 (cf. [4,5,22]).** *Let $M = (X_0, X_1, \ldots)$ be an ergodic Markov chain on finite space $\mathcal{X}$ with transition matrix $P$ and stationary distribution $\pi$.*

- *For any two probability distributions $\mu$ and $\nu$ on space $\mathcal{X}$ the relative entropy $D(\mu P^t || \nu P^t)$ decreases with $t$, i.e. $D(\mu P^t || \nu P^t) \ge D\left(\mu P^{t+1} || \nu P^{t+1}\right)$.*
- *For any initial distribution $\mu$ the relative entropy $D(\mu P^t||\pi)$ decreases with $t$. Furthermore, $\lim_{t\to\infty} D(\mu P^t||\pi) = 0$.*
- *The conditional entropy $H(X_0|X_t)$ is increasing in $t$.*

## References


1. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: FOCS, pp. 218–223 (1979)
2. Alon, N., Avin, C., Koucký, M., Kozma, G., Lotker, Z., Tuttle, M.R.: Many random walks are faster than one. Comb. Probab. Comput. **20**(4), 481–502 (2011)
3. Boyd, S.P., Diaconis, P., Xiao, L.: Fastest mixing Markov chain on a graph. SIAM Rev. **46**(4), 667–689 (2004)
4. Cover, T.M.: Which processes satisfy the second law. In: Halliwell, J.J., Pérez-Mercader, J., Zurek, W.H. (eds.) PhysicaL Origins of Time Asymmetry, pp. 98–107 (1994)
5. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley, Hoboken (2006)
6. Díaz, C.: Anonymity metrics revisited. In: Anonymous Communication and Its Applications, 09–14 October 2005 (2005)
7. Dragomir, S., Scholz, M., Sunde, J.: Some upper bounds for relative entropy and applications. Comput. Math. Appl. **39**(9), 91–100 (2000)
8. Efremenko, K., Reingold, O.: How well do random walks parallelize? In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX/RANDOM - 2009. LNCS, vol. 5687, pp. 476–489. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03685-9_36
9. Elsässer, R., Sauerwald, T.: Tight bounds for the cover time of multiple random walks. Theor. Comput. Sci. **412**(24), 2623–2641 (2011)
10. Feige, U.: A tight lower bound on the cover time for random walks on graphs. Random Struct. Algorithms **6**(4), 433–438 (1995)
11. Feige, U.: A tight upper bound on the cover time for random walks on graphs. Random Struct. Algorithms **6**(1), 51–54 (1995)
12. Gao, K., Zhu, Y., Gong, S., Tan, H.: Location privacy protection algorithm for mobile networks. EURASIP J. Wirel. Commun. Netw. **2016**(1), 205 (2016)
13. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov Chains and Mixing Times. AMS, Providence (2009)
14. Li, M., Zhu, H., Gao, Z., Chen, S., Yu, L., Hu, S., Ren, K.: All your location are belong to us: breaking mobile social networks for automated user location tracking. In: MobiHoc, pp. 43–52. ACM, New York (2014)
15. Lovász, L.: Random walks on graphs: a survey. Comb. Paul Erdos Eighty **2**(1), 1–46 (1993)
16. Niroj Kumar Pani, B.K.R., Mishra, S.: A topology-hiding secure on-demand routing protocol for wireless ad hoc network. Int. J. Comput. Appl. **144**(4), 42–50 (2016)
17. Nonaka, Y., Ono, H., Sadakane, K., Yamashita, M.: The hitting and cover times of metropolis walks. Theor. Comput. Sci. **411**(16–18), 1889–1894 (2010)
18. Pfitzmann, A., Köhntopp, M.: Anonymity, unobservability, and pseudonymity — a proposal for terminology. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 1–9. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44702-4_1
19. Pham, A., Huguenin, K., Bilogrevic, I., Hubaux, J.P.: Secure and private proofs for location-based activity summaries in urban areas. In: UbiComp, pp. 751–762. ACM, New York (2014)

20. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3rd edn. Cambridge University Press, New York (2007)
21. Reingold, O.: Undirected connectivity in log-space. J. ACM **55**(4), 17 (2008)
22. Rényi, A.: On measures of entropy and information. In: Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 547–561. University of California Press (1961)
23. Sun, J., Boyd, S.P., Xiao, L., Diaconis, P.: The fastest mixing markov process on a graph and a connection to a maximum variance unfolding problem. SIAM Rev. **48**(4), 681–699 (2006)
24. Zhang, Y., Yan, T., Tian, J., Hu, Q., Wang, G., Li, Z.: TOHIP: a topology-hiding multipath routing protocol in mobile ad hoc networks. Ad Hoc Netw. **21**, 109–122 (2014)

# Probabilistic Algorithms

# Parallel Search with No Coordination

Amos Korman[1(✉)] and Yoav Rodeh[2]

[1] CNRS, University Paris Diderot, Paris, France
amos.korman@irif.fr
[2] Weizmann Institute of Science, Rehovot, Israel

**Abstract.** We consider a parallel version of a classical Bayesian search problem. $k$ agents are looking for a treasure that is placed in one of the boxes indexed by $\mathbb{N}^+$ according to a known distribution $p$. The aim is to minimize the expected time until the first agent finds it. Searchers run in parallel where at each time step each searcher can "peek" into a box. A basic family of algorithms which are inherently robust is *non-coordinating* algorithms. Such algorithms act independently at each searcher, differing only by their probabilistic choices. We are interested in the price incurred by employing such algorithms when compared with the case of full coordination.

We first show that there exists a non-coordination algorithm, that knowing only the relative likelihood of boxes according to $p$, has expected running time of at most $10 + 4(1 + \frac{1}{k})^2 T$, where $T$ is the expected running time of the best fully coordinated algorithm. This result is obtained by applying a refined version of the main algorithm suggested by Fraigniaud, Korman and Rodeh in STOC'16, which was designed for the context of linear parallel search.

We then describe an optimal non-coordinating algorithm for the case where the distribution $p$ is known. The running time of this algorithm is difficult to analyse in general, but we calculate it for several examples. In the case where $p$ is uniform over a finite set of boxes, then the algorithm just checks boxes uniformly at random among all non-checked boxes and is essentially 2 times worse than the coordinating algorithm. We also show simple algorithms for Pareto distributions over $M$ boxes. That is, in the case where $p(x) \sim 1/x^b$ for $0 < b < 1$, we suggest the following algorithm: at step $t$ choose uniformly from the boxes unchecked in $\{1, \ldots, \min(M, \lfloor t/\sigma \rfloor)\}$, where $\sigma = b/(b + k - 1)$. It turns out this algorithm is asymptotically optimal, and runs about $2 + b$ times worse than the case of full coordination.

## 1 Introduction

We consider a parallel variant of the classical Bayesian search problem, typically attributed to Blackwell [6]. A treasure is placed in one of the boxes indexed by $\mathbb{N}^+$

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 648032).


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 195–211, 2017.
https://doi.org/10.1007/978-3-319-72050-0_12

according to some known distribution $p$. As $p$ is known, we can assume that the boxes are ordered so that $p$ is non-increasing. Denote $M = \max\{x \mid p(x) > 0\}$, which can be $\infty$. There are $k$ agents that search for the treasure, aiming to minimize the expected time until the first one finds it, where looking into a box takes one unit of time. We shall assume that algorithms know the number of searchers $k$.

If coordination is allowed, a simple application of the rearrangement inequality shows that letting agent $i$ peek into box $(t-1)k+i$ at time $t$ is an optimal algorithm (a formal proof can be found in [20]). Denote this algorithm $\mathtt{A}_{cord}$, and note that its expected running time is $\sum_x p(x)\lceil x/k \rceil$, giving a speedup of essentially $k$ compared to just one searcher. However, as simple as this algorithm is, it is very sensitive to faults of all sorts. For example, if one searcher crashes at some point during the execution then the searchers may completely miss the treasure, unless the protocol employs some mechanism for detecting such faults[1].

A class of search algorithms which is of particular interest is *non-coordinating* algorithms [1,17]. In such an algorithm, all searchers operate independently, executing the same protocol, differing only in the outcome of the flips of their random coins. With such a strong restriction on the coordination, one cannot expect that many search problems could be efficiently parallelized. However, when such a parallelization can be achieved, the benefit can potentially be high, not only in terms of saving in communication and overhead in computation, but also in terms of robustness. To get some intuition, assume that an oblivious adversary is allowed to crash at most $f$ out of the $k$ searchers at arbitrary points in time during the execution. To overcome the presence of at most $f$ faults, one can simply run the non-coordinating algorithm that is designed for the case of $k-f$ searchers. If the running time of a non-coordinating algorithm without crashes is $T(k)$, then the running time of the new robust algorithm would be at most $T(k-f)$. This is because the correct operation as well as the running time of a non-coordinating algorithm can only improve if more searchers than planned are actually being used. Note that even when coordination is allowed, one cannot expect to obtain robustness at a cheaper price since the number of searchers that remain alive is in the worst case $k-f$.

For an algorithm $A$, and $k \geq 2$, denote by $\mathbb{T}_k(A,x)$ the expected running time if the treasure is placed at $x$, when running algorithm $A$ with $k$ searchers. Note that by "running time" we actually mean the expected number of boxes peeked into by each searcher, as we are mostly interested in query complexity. Further, for a distribution $p$ over the boxes, denote the expected time to find the treasure when it is placed in one of the boxes according to $p$:

$$\mathbb{T}_{p,k}(A) = \sum_x p(x)\mathbb{T}_k(A,x)$$

[1] It is actually an interesting and non trivial question to find efficient and robust algorithms that are allowed to coordinate [8]. Of course, our non-coordinating algorithms fall under this category, but one may potentially improve the running time by allowing coordination.

In this notation, the expected running time of the optimal coordinating algorithm is $\mathbb{T}_{p,k}(\mathtt{A}_{cord})$. We are interested in the connection between these two terms, and specifically in identifying non-coordinating algorithms that minimize the additive and multiplicative factors $a$ and $b$ such that:

$$\mathbb{T}_{p,k}(A) \leq a + b\mathbb{T}_{p,k}(\mathtt{A}_{cord})$$

We remark, for readability's sake, the subscripts above, as well as most subscripts in the text that follows, will be dropped when clear from context. Also, the number of agents $k \geq 2$ will be fixed and so omitted from formal statements. This will many times go for $p$ as well. Also note that there are distributions where no algorithm can achieve finite running time, such as $p(x) = c/x^2$, where the expected placement of the treasure is unbounded. We shall therefore always assume that $\sum_x p(x)x < \infty$, and so, for example, $\mathbb{T}(\mathtt{A}_{cord})$ is always defined.

### 1.1 Our Results

We first show that there exists a simple and highly efficient algorithm, denoted $\mathtt{A}_{universal}$, which for large $k$ enjoys a multiplicative factor that tends to 4. In this algorithm, each agent, at phase $t$, checks two different uniformly chosen boxes of those it did not check yet in $\{1, \ldots, t(k+1)\}$. This algorithm is universal in the sense that it does not depend on the details of the distribution $p$, and assumes only the knowledge of the relative likelihood of the boxes, that is, their order.

**Theorem 1.** $\mathbb{T}(\mathtt{A}_{universal}) \leq 10 + 4\left(1 - \frac{1}{k+1}\right)^2 \mathbb{T}(\mathtt{A}_{cord})$

Note that this gives improvement over the trivial one searcher for every $k$. Even for $k = 2$ we get that for large enough $x$, this runs at 8/9's the time of the lone searcher.

Algorithm $\mathtt{A}_{universal}$ remembers all the boxes it checked and so needs memory which is linear in its running time. We also consider $\mathtt{A}_{memory}$ which at phase $t$ chooses uniformly two boxes in $\{1, \ldots, tk\}$. This algorithm uses only logarithmic memory in its running time, and for large number of searchers performs almost as well:

**Theorem 2.** $\mathbb{T}(\mathtt{A}_{memory}) \leq 2 + 4\mathbb{T}(\mathtt{A}_{cord})$

Both algorithms $\mathtt{A}_{universal}$ and $\mathtt{A}_{memory}$ where actually given in [17] to tackle the setting of linear search with an adversarially placed treasure. We note, however, that when applied in our context, the bounds established in [17] only guarantee that the additive term is some unknown, possibly large constant. To prove that this constant is in fact small we had to refine the upper bound analysis of [17], and prove tighter bounds on the Gamma function.

We next present Algorithm $\mathtt{A}^\star$, that given access to the exact distribution $p$ (and not only the order of the boxes), gives the optimal expected running time:

**Theorem 3.** *For every non-coordinating algorithm $A$, $\mathbb{T}(\mathtt{A}^\star) \leq \mathbb{T}(A)$.*

An interesting property satisfied by this algorithm, is that at any time during the execution, all boxes that previously received a positive probability to be checked, are now going to be checked with equal probability.

Calculating the running time of $\mathtt{A}^\star$ can become challenging for specific distributions, and the rest of the paper shows a few interesting examples. A simple one is when $p$ is the uniform distribution over a finite domain. In this case, running $\mathtt{A}^\star$, at each step each agent chooses a box uniformly among those it did not check yet. This natural choice for an algorithm therefore turns out to be optimal, and yields a multiplicative factor of essentially 2 when compared to $\mathtt{A}_{cord}$ (when the number of searchers is large).

On the other extremity there are exponential distributions. Such distributions strongly concentrate the probability on the first few boxes, and so a good algorithm would invest in optimizing the parallel performance on a constant number of boxes. As we are concerned with non-trivial behavior over many boxes, we turn our attention to investigate Pareto distribution, which spread the distribution more gradually.

Specifically, we consider the family of Pareto distributions over $M$ boxes, thinking of $M$ as large. Here, for some $0 < b < 1$, for all $x \leq M$, $p(x) = I/x^b$, where $I$ is the normalization factor, and $p(x) = 0$ for larger $x$. While $\mathtt{A}^\star$ is optimal, it is quite complex and difficult to analyse. We present a simple algorithm $\mathtt{A}_{pareto}$ that is asymptotically optimal. In $\mathtt{A}_{pareto}$, at step $t$, an agent chooses uniformly from one of the boxes it did not check yet in $\{1, \ldots, \min(M, \lfloor t/\sigma \rfloor)\}$, where $\sigma = b/(b + k - 1)$.

**Theorem 4.** *For* $0 < b < 1$,

$$\lim_{M \to \infty} \frac{\mathbb{T}_{r_{M,b}}(\mathtt{A}_{pareto})}{\mathbb{T}_{r_{M,b}}(\mathtt{A}_{cord})} = k\sigma(2-\sigma) + \frac{k}{k+1}(2-b)(1-\sigma)^2.$$

*Furthermore, no non-coordinating algorithm can achieve a better limit bound.*

When $b$ is close to 1, then $\sigma \approx 1/k$ and the factor becomes $(3k-1)/(k+1)$. For $k = 2$ this is $5/3$ compared to $16/9$ achieved by $\mathtt{A}_{universal}$, and for large $k$ this tends to 3 as opposed to 4. For smaller $b$'s the result is not as clean, but assuming $k$ is large, then $\sigma \approx b/k$, and we get that the ratio is about $2 + b$. This makes sense, as when $b$ approaches 0, the distribution becomes uniform, where we already know that this factor is 2 for large $k$.

Finally, we note that most of our algorithms are very simple and hence applicable. From the technical point of view, our results illustrate deep connections between the general probabilistic parallel search setting considered here, and the setting of parallel *linear* search studied in [17].

### 1.2 Related Work

The study of parallel search by non-coordinating algorithms has recently been advocated by Fraigniaud et al. as a simple way to obtain robustness while avoiding communication overheads [17]. The setting therein, however, differs from

ours by two fundamental characteristics: First, they assumed that the treasure is placed by an adversary. The second major difference is that they focused on a *linear search* setting (see also [4,5,9]), in which the boxes are linearly ordered and the objective is to find a treasure placed in a box in time that is compared to its index. That is, if the treasure is placed in index $x$, then the running time of the parallel algorithm should be compared to $x/k$. Although this linear search setting may seem somewhat specific compared to the setting studied in the current paper, it turns out that there are important connections between the two settings, both in terms of techniques and results. See Sect. 2 for more details.

The case of a single searcher that searches for a randomly pleased treasure has receives significant amount of attention from the communities of statistics, operational research and computer science, see e.g., [6,11,21], and has been studied under various settings, including the case that there are different costs associated with queries, that queries can be noisy, and that the target may be mobile, see the book [23]. As we initiate its parallel version, we consider only the most basic form of the problem, yet, we note that most of our results can be extended to the case in which weighted costs are associated with queries.

In general, when it comes to parallel search, most of the literature deals with mobile agents that search graphs of different topologies, and typically employ some form of communication between themselves. The literature on this subject is vast, and some good references can be found, e.g., in [2,3,10,16,22]. The major difference between our setting and the mobile agent setting, is that we allow "random access" to the different boxes. That is, our searcher can jump between different boxes at no cost. In other words, our focus is on the *query complexity* rather than the *move complexity*.

Multiple random walkers are a special case of non-coordinating searchers. In a series of papers [1,7,12,13] several results regarding hitting time, cover time and mixing times are established, such as a linear speedup for several graph families including expanders and random graphs. Non-coordinating searchers have also been studied in the context of the ANTS problem, a parallel variant of the cow-path problem on the grid [4,19], which was introduced in [14,15] motivated by applications to central search foraging by desert ants. For example, it was shown in [14,15] that a speedup of $O(k)$ can be achieved with $k$ non-coordinating searchers, and that a linear speedup cannot be achieved unless the agents have some knowledge of $k$.

Finally, BOINC [18] (Berkeley Open Infrastructure for Network Computing) is a platform for volunteer computing supporting dozens of projects including the famous SETI@home analyzing radio signals for identifying signs of extra terrestrial intelligence. Most projects maintained at BOINC use parallel search mechanisms where a central server controls and distributes the work to volunteers. The framework in this paper is a potential abstraction for projects operated at platforms similar to BOINC with hundreds of thousands distributed searchers.

## 2 Ordering of Boxes is Known

In [17], the authors consider a somewhat different scenario. The boxes are ordered linearly by some predefined importance, and the treasure is placed in one of them by an adversary. In such a situation, a lone searcher will check the boxes according to their order, and so box $x$ will be checked by time $x$. They present algorithm $\mathtt{A}_{universal}$, in which each agent, at phase $t$, checks two different uniformly chosen boxes of those it did not check yet in $\{1, \ldots, t(k+1)\}$. It is shown there that:

$$\limsup_{x \to \infty} \frac{\mathbb{T}(\mathtt{A}_{universal}, x)}{x} = \frac{4k}{(k+1)^2}$$

and that it is in fact optimal in this way. That is, in that setting, it has the best speedup compared to the lone searcher when taking large enough $x$.

If $\mathtt{A}_{universal}$ would give this result for all $x$ and not only large ones, it will solve the case of a randomly placed treasure with surprising efficiency. All one has to do is set the importance of the boxes according to the likelihood of the treasure being placed there.

The proof of the next lemma follows a refined analysis of that done in [17], and shows that the limsup only hides a small additive term:

**Lemma 1.** *For all $x$, $\mathbb{T}(\mathtt{A}_{universal}, x) \leq 10 + \frac{4k}{(k+1)^2} x$.*

A major ingredient in the proof is the following lemma, whose proof appears in [20].

**Lemma 2.** *For integers $b \geq a \geq 1$, and $0 < \phi \leq 1$, $\prod_{i=a}^{b} \frac{i}{i+\phi} \leq \left(\frac{a}{b}\right)^{\phi}$.*

Using properties of the Gamma function it is easy to see that the two sides of the equation are asymptotically equal, but this is not enough to prove our result as we need the inequality for small $a$ and $b$ as well.

*Proof (of Lemma 1).* Count the time in steps of size 2, so at each step $\mathtt{A}_{universal}$ chooses two new boxes. The algorithm might actually end mid-step, but this just means that this is an over approximation.

The number of elements the algorithm chooses from at step $t$ is $(k+1)t - 2(t-1) = (k-1)t + 2$. Box $x$ starts to have some probability of being checked at time $s = \lceil x/(k+1) \rceil$, and for $t \geq s$ the probability of $x$ not being checked by time $t$ is:

$$\prod_{i=s}^{t} \left(1 - \frac{2}{(k-1)i+2}\right)^k = \prod_{i=s}^{t} \left(\frac{(k-1)i}{(k-1)i+2}\right)^k = \prod_{i=s}^{t} \left(\frac{i}{i+\frac{2}{k-1}}\right)^k \leq \left(\frac{s+1}{t}\right)^{\frac{2k}{k-1}}$$

where the last step is by Lemma 2. Denoting $a = 2k/(k-1)$ the total running time for $x$ is then at most (times 2):

$$s + 2 + \sum_{t=s+2}^{\infty} \left(\frac{s+1}{t}\right)^a$$

As $((s+1)/t)^a$ is decreasing, we can bound the sum from above by taking the integral but starting it at $s+1$ and not $s+2$. This gives the upper bound of:

$$s+2+\int_{s+1}^{\infty}\left(\frac{s+1}{t}\right)^a \mathrm{d}t = s+2+(s+1)\int_1^{\infty} t^{-a}\,\mathrm{d}t = s+2+\frac{s+1}{a-1}$$
$$= 1+(s+1)\left(1+\frac{1}{\frac{2k}{k-1}-1}\right) = 1+\left(\left\lceil\frac{x}{k+1}\right\rceil+1\right)\left(1+\frac{k-1}{k+1}\right)$$
$$\leq 1+\left(\frac{x}{k+1}+2\right)\left(\frac{2k}{k+1}\right) \leq 5+\frac{2k}{(k+1)^2}s$$

Multipying by 2 gives the result. □

Using Lemma 1 the proof of Theorem 1 becomes straightforward, as follows:

*Proof.*

$$\mathbb{T}(\mathtt{A}_{universal}) = \sum_x p(x)\mathbb{T}_k(\mathtt{A}_{universal},x) \leq 10+\frac{4k}{(k+1)^2}\sum_x p(x)x$$
$$\leq 10+\frac{4k^2}{(k+1)^2}\sum_x p(x)\left\lceil\frac{x}{k}\right\rceil = 10+4\left(1-\frac{1}{k+1}\right)^2\mathbb{T}(\mathtt{A}_{cord})$$

□

At [17], the authors introduce a memory efficient version of $\mathtt{A}_{universal}$, which we present here, slightly altered, as $\mathtt{A}_{memory}$. In it, each agent, at phase $t$, checks two uniformly chosen boxes of those in $\{1,\ldots,kt\}$.

**Lemma 3.** *For all $x$, $\mathbb{T}(\mathtt{A}_{memory},x) \leq 2+4\left\lceil\frac{x}{k}\right\rceil$.*

*Proof.* The proof proceeds in very similar to that of Lemma 1, yet is in fact a little simpler. Count the time in steps of size 2.

Box $x$ starts to have some probability of being checked at time $s=\lceil x/k\rceil$, and for $t\geq s$ the probability of $x$ not being checked by time $t$ is:

$$\prod_{i=s}^{t}\left(\left(1-\frac{1}{ki}\right)^2\right)^k \leq \prod_{i=s}^{t}\left(1-\frac{1}{ki+1}\right)^{2k} = \prod_{i=s}^{t}\left(\frac{i}{i+\frac{1}{k}}\right)^{2k} \leq \left(\frac{s}{t}\right)^2$$

where the last step is by Lemma 2. The total running time for $x$ is then at most (times 2):

$$s+1+\sum_{t=s+1}^{\infty}\left(\frac{s}{t}\right)^2$$

As $(s/t)^2$ is decreasing, we can bound the sum from above by taking the integral but starting it at $s$ and not $s+1$. This gives the upper bound of:

$$s+1+\int_s^{\infty}\left(\frac{s}{t}\right)^2\mathrm{d}t = s+1+s\int_1^{\infty}\frac{1}{t^2}\,\mathrm{d}t = 2s+1$$

Multipying by 2 gives the result. □

Note that for $k \leq 4$ Lemma 3 is of no use, as running the trivial one searcher will do better.

Lemma 3 immediately proves Theorem 2. While $\mathtt{A}_{memory}$ is not optimal as $\mathtt{A}_{universal}$ is, as $k$ grows the difference between them grows smaller, and $\mathtt{A}_{memory}$'s simplicity and efficiency make it an outstanding candidate for real life purposes.

## 3 Exact Distribution is Known

Ignoring the small additive term in Theorem 1, as $k$ grows larger we get that $\mathtt{A}_{universal}$ is about 4 times worse than the best coordinating algorithm. In the remainder of the paper we show it is possible to improve on this if the exact distribution is known.

### 3.1 Preliminaries

Consider a non-coordinated algorithm $A$ that is running on $k$ agents. Focusing on just one agent, denote by $A(x,t)$ the probability that by time $t$, box $x$ was not already checked by this agent. Hence, the probability that none of the $k$ agents checked $x$ by time $t$ is $A(x,t)^k$. In fact, as we shall soon see, the information encoded in this functional view of $A$ is all that is needed to assess its running time. First note:

**Observation 5.** *The function corresponding to algorithm $A$ satisfies $A(x,0) = 1$ for all $x$. Also, for all $x$ and $t \geq 1$:*

$$A(x,t) = A(x,t-1) \cdot \mathtt{Pr}\left[\begin{array}{l|l} x \text{ wasn't checked} & x \text{ wasn't checked} \\ \text{at time } t & \text{prior to time } t \end{array}\right]$$

Let us now consider such functions on their own, possibly without a corresponding algorithm. Let[2] $N : \mathbb{N}^+ \times \mathbb{N} \to [0,1]$. For time $t$, denote:

$$C_N(t) = \sum_x 1 - N(x,t)$$

In the case of an algorithm $A$, $C_A(t)$ is the expected number of elements that were checked by time $t$ by just one of the searchers running $A$, and is therefore at most $t$. We say that $N$ satisfies the *column requirement* at time $t$ if $C_N(t) \leq t$. Also, define the set of *valid* functions as:

$$\mathcal{V} = \left\{ N : \mathbb{N}^+ \times \mathbb{N} \to [0,1] \,\middle|\, \forall t, C_N(t) \leq t \right\}$$

and so functions corresponding to algorithms are always valid. Finally, the "running time" of $N$:

$$\mathbb{T}_{p,k}(N) = \sum_x p(x) \sum_t N(x,t)^k = \sum_t \sum_x p(x) N(x,t)^k$$

[2] The letter $N$ stands for "probability of *not* being checked up to time".

The sum on $t$ is from 0 to $\infty$, and these limits will be omitted whenever clear from context. This is clearly defined so that $\mathbb{T}(A)$ is indeed the expected running time of algorithm $A$, as

$$\mathbb{T}(A,x) = \sum_t \mathtt{Pr}\left[x \text{ wasn't found by time } t\right] = \sum_t A(x,t)^k.$$

To lower bound the running time of algorithms, we find the optimal $N \in \mathcal{V}$, in the sense that it minimizes $\mathbb{T}(N)$. For that, we introduce a generalized version of the main Lemma of [17] which we prove in [20]. At this point we only need a very simple version of the lemma, yet we present it in its full glory, as we will need it later in the paper. The current version improves on the original lemma of [17] as it applies to general measurable functions, instead of only continuous and bounded ones. In addition, the measure theoretic proof is much more elegant and concise than the original one.

### 3.2 Main Lemma

The notation that follows is in measure theory style. Fix some $k \geq 2$ and let $(X, \mathcal{X}, \mu)$ be a measure space. For $T \geq 0$, denote by $V(T)$ the set of measurable functions $f : X \to [0,1]$ such that $\int 1 - f \, \mathrm{d}\mu \leq T$. For a measurable function $c : X \to [0,\infty)$, and $\alpha \geq 0$ define the function $f_{c,\alpha} : X \to [0,1]$ as:

$$f_{c,\alpha}(x) = \begin{cases} 1 & c(x) = 0 \\ \min\left(1, \alpha c(x)^{-\frac{1}{k-1}}\right) & \text{otherwise} \end{cases}$$

**Lemma 4.** *For a given $c$ and $T$ as above, if there is some $h \in V(T)$ such that $\int ch^k \, \mathrm{d}\mu < \infty$, then there exists $\alpha \geq 0$, such that $f_{c,\alpha} \in V(T)$, and for every $g \in V(T)$, $\int c f_{c,\alpha}^k \, \mathrm{d}\mu \leq \int cg^k \, \mathrm{d}\mu$. Furthermore, this $\alpha$ is minimal among those satisfying $f_{c,\alpha} \in V(T)$.*

Towards finding the optimal $N \in \mathcal{V}$, fix some $t$, and then $N \in \mathcal{V}$, means $\sum_x 1 - N(x,t) \leq t$, and the aim is to minimize $\sum_x p(x)N(x,t)^k$. As this can be done for each $t$ completely separately, Lemma 4 comes into play.

**Lemma 5.** *The following function $L$ is in $\mathcal{V}$, and achieves minimal $\mathbb{T}(\cdot)$ over all valid functions.*

$$L_{p,k}(x,t) = \begin{cases} 1 & p(x) = 0 \\ \min(1, \alpha(t)q(x)) & \text{otherwise} \end{cases}$$

*where $q(x) = p(x)^{-\frac{1}{k-1}}$, and for all $t$, $\alpha(t) \geq 0$ is the minimal such that $L_{p,k} \in \mathcal{V}$.*

*Proof.* Fix $t$. Setting $X = \mathbb{N}^+$ with the trivial measure $\mu(x) = 1$ for all $x$, $T = t$ and $c = p$, Lemma 4 gives the values of the optimal $N$ for this specific $t$. To check the condition of the lemma, take the constant function $h(x) = 1$. Clearly $h \in V(t)$, and $\int ch^k \, \mathrm{d}\mu = \sum_x p(x) = 1 < \infty$. □

The following basically says that $L$, if thought of as an algorithm, never rechecks a box.

**Observation 6.** *For every $t < M$, $C_L(t) = t$, and for $t \geq M$, $L(x,t) = 0$ everywhere.*

*Proof.* For $t \geq M$, $\alpha(t) = 0$ satisfies the column requirement, and is as required. Next assume that $0 < t < M$. Clearly, in this case $\alpha(t) \neq 0$, as otherwise the column requirement is violated. Assume by contradiction that $C_L(t) \neq t$, and since $L$ is valid this means that $C_L(t) < t$.

Note that since $p(x)$ goes to zero, $q(x)$ goes to infinity, and so there are only a finite number of $x$'s where $\alpha(t)q(x) < 2$. As $\alpha(t) > 0$, we can reduce it slightly, and this will only affect the value of $L$ at these $x$'s. Making this change small enough, will maintain the inequality $C_L(t) < t$, and keep $L$ valid. As this change can only decrease $L(x,t)$ at these points, $\mathbb{T}(L)$ does not increase. Contradicting the minimality of $\alpha(t)$.

As an illustration consider a simple example: $k = 2$, $p(1) = 1/2$, $p(2) = 1/3$, and $p(3) = 1/6$. In this case, $q(1) = 2$, $q(2) = 3$ and $q(3) = 6$, and some quick calculations show that $\alpha(1) = 1/5$, $\alpha(2) = 1/11$, and $\alpha(3) = 0$. From these we get the matrix $L$ on the right. Note that Observation 6 holds, as the sum of column $t$ is indeed equal to $M - t$.

| $t\rightarrow$ / $x\downarrow$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 1 | 1 | $2/5$ | $2/11$ | 0 |
| 2 | 1 | $3/5$ | $3/11$ | 0 |
| 3 | 1 | 1 | $6/11$ | 0 |

### 3.3 Optimal Algorithm

Although it may seem that every valid function $N$ has a corresponding algorithm, it is not at all clear, because the conditional probabilities arising from Observation 5 quickly become complicated for general $N$. However, it turns out that because of the specific structure $L$ has, there is in fact an algorithm that has it as its function.

For instance, a corresponding algorithm for the example above is: (1) choose box 1 w.p. 0.6, and otherwise choose box 2. (2) choose box 3 w.p. 5/11, and otherwise the unchosen box of 1 and 2. (3) choose the last remaining box. Note especially step (2), where the remaining probability of 6/11 is used to check the unchosen box $B$ from 1 and 2, and indeed, by Observation 5, $(2/11)/0.4 = (3/11)/0.6 = 5/11$, which is the probability of not checking $B$ given that it was not checked up to this point.

In this section we present Algorithm $\mathtt{A}^\star$, which given $p$, calculates the function $L$, and randomly chooses boxes so as to get $L$ as its function. Based on Lemma 5, this will show that $\mathbb{T}(\mathtt{A}^\star) \leq \mathbb{T}(A)$ for every non-coordinating algorithm $A$, and hence establish Theorem 3.

We next provide an intuitive explanation for why $\mathtt{A}^\star$ implements $L(x,t)$. At step $t$, the first thing $\mathtt{A}^\star$ does is calculate the values of $L(x,t)$ for all $x$, so that it

**Algorithm** $\mathtt{A}^\star$

$ac(0) \leftarrow 0$
**for** $t \leftarrow 1$ to $M$ **do**
  $y \leftarrow \mathtt{ac}(t-1)$ ▷ Calculate $\mathtt{ac}(t)$
  **while** $y < M$ **and** $\sum_{x=1}^{y+1} 1 - q(x)/q(y) \leq t$ **do**
    $y \leftarrow y + 1$
  $\mathtt{ac}(t) \leftarrow y$
  $\alpha(t) \leftarrow (\mathtt{ac}(t) - t)/\sum_{x \leq \mathtt{ac}(t)} q(x)$ ▷ Calculate $\alpha(t)$
  **from** unchecked boxes $x \leq \mathtt{ac}(t)$ ▷ Choose one box
    **if** $x \leq \mathtt{ac}(t-1)$ **then**
      Check $x$ w.p. $1 - \alpha(t)/\alpha(t-1)$
    **else**
      Check $x$ w.p. $1 - \alpha(t)q(x)$

can recreate them with its random choices. For that it needs to calculate $\alpha(t)$, which by Observation 6 means solve the equation:

$$t = \sum_x 1 - L(x,t) = \sum_x 1 - \min(1, \alpha(t)q(x)) \tag{1}$$

The first step is to figure out which $x$'s actually contribute something to this sum. Say box $x$ is *active* at time $t$ if $L(x,t) < 1$. As $L$ is non-decreasing in $x$, there is some $\mathtt{ac}(t)$, s.t. the set of active boxes at time $t$ is $\{1, \ldots, \mathtt{ac}(t)\}$. To calculate $\mathtt{ac}(t)$, $\mathtt{A}^\star$ gradually decreases $\alpha(t)$, while keeping the column requirement satisfied. The point is, $x$ is active when $\alpha(t) < 1/q(x)$, and so to see who is active, it needs to only check $\alpha(t) = 1/q(1), 1/q(2), \ldots$. Once $\mathtt{ac}(t)$ is found, solving (1) and finding $\alpha(t)$ is straightforward.

Now that $L(x,t)$ is calculated, $\mathtt{A}^\star$ randomly chooses a box to check according to it, using the fact that up to this point, the probability that box $x$ was not checked is $L(x, t-1)$. If a box was not active, and now is, then clearly it should be checked with probability $1 - q(x)\alpha(t)$. If it was already active, then it should change from $q(x)\alpha(t-1)$ to $q(x)\alpha(t)$, which by Observation 5 means it should be checked with probability $1 - \alpha(t)/\alpha(t-1)$. Fortunately, all these probabilities sum up to 1.

We next formalize the aforementioned intuitive description thus establishing the optimality of $\mathtt{A}^\star$.

*Proof (of Theorem 3).* As mentioned, based on Lemma 5, it is enough to prove that indeed $\mathtt{A}^\star$ implements $L(x,t)$. First, $\mathtt{A}^\star$ calculates $\alpha(t)$ and $\mathtt{ac}(t)$. Note that $y \leq \mathtt{ac}(t)$ iff $\alpha(t) < 1/q(y)$, and so, to calculate $\mathtt{ac}(t)$, it is enough to check values for $\alpha(t)$ that are equal to $1/q(y)$ for $y > \mathtt{ac}(t-1)$. Once we know $\mathtt{ac}(t)$, by Observation 6:

$$t = \sum_{x \leq \mathtt{ac}(t)} 1 - \alpha(t)q(x)$$

Solving this for $\alpha(t)$ is what the algorithm does.

To show that the next part of $\mathtt{A}^\star$ is at all valid, we show that the probabilities of each step add up to at most 1. The number of boxes that were already active at $t-1$, and were not checked yet at time $t$ is $\mathtt{ac}(t-1)-(t-1)$. So, summing all the probabilities of the different boxes:

$$(\mathtt{ac}(t-1)-t+1)\left(1-\frac{\alpha(t)}{\alpha(t-1)}\right)+\sum_{\mathtt{ac}(t-1)<x\leq\mathtt{ac}(t)} 1-\alpha(t)q(x) \qquad (2)$$

By Observation 6:

$$\sum_{x\leq\mathtt{ac}(t-1)} 1-\alpha(t-1)q(x)=t-1 \quad\Longrightarrow\quad \sum_{x\leq\mathtt{ac}(t-1)} \alpha(t-1)q(x)=\mathtt{ac}(t-1)-t+1$$

Plugging this is (2):

$$\begin{aligned}&\sum_{x\leq\mathtt{ac}(t-1)}(\alpha(t-1)-\alpha(t))q(x)+\sum_{\mathtt{ac}(t-1)<x\leq\mathtt{ac}(t)} 1-\alpha(t)q(x)\\ &=\sum_{x\leq\mathtt{ac}(t)} 1-\alpha(t)q(x)-\sum_{x\leq\mathtt{ac}(t-1)} 1-\alpha(t-1)q(x)\end{aligned}$$

By Observation 6 the first sum is $t$ and the second is $t-1$, and so the sum of probabilities is indeed 1.

The last bit is to show that indeed $\mathtt{A}^\star=L$. This is proved by induction on $t$. For $t=0$, $L(x,1)=\mathtt{A}^\star(x,1)$ for all $x$. Assume equality for $t-1$ and we prove it for $t$. For $x\leq\mathtt{ac}(t-1)$, $\mathtt{A}^\star(x,t-1)=L(x,t-1)=\alpha(t-1)q(x)$. Using Observation 5:

$$\mathtt{A}^\star(x,t)=\mathtt{A}^\star(x,t-1)\cdot\frac{\alpha(t)}{\alpha(t-1)}=\alpha(t-1)q(x)\cdot\frac{\alpha(t)}{\alpha(t-1)}=L(x,t)$$

For $\mathtt{ac}(t-1)<x\leq\mathtt{ac}(t)$, it is straightforward. □

As an interesting side note, observe that at each step, all previously active yet unchecked boxes get the same probability of being checked. Moreover, this probability does not depend at all at the previous choices made by the algorithm. This point sounds counter-intuitive from a Bayesian point of view, as we would expect a rescaling of the probabilities that differs according to the history we've already seen.

An important point is that $\mathtt{A}^\star$ has at each step a finite set of boxes to choose from. As $p$ goes to 0, $q$ goes to infinity, and so if there are an infinite number of active boxes, then $\alpha$ must be 0, but that means that all boxes were surely checked.

How does algorithm $\mathtt{A}^\star$ look for example distributions, and how does it compare to $\mathtt{A}_{universal}$? In general it is quite difficult to analyse the exact running time of this algorithm, but sometimes it can be done, as we shall see.

### 3.4 Uniform Distribution

The first example that comes to mind is when the treasure is uniformly placed in one of the boxes $\{1, \ldots, M\}$. As $q(x)$ is equal for all boxes, an agent running $\mathtt{A}^\star$ will at the first step choose among them uniformly, and continue to do so at each step, choosing from those that it did not check yet. This algorithm is the most natural choice in this case, and indeed, by Theorem 3 it is optimal. Analysis is simple, and we do approximate it here for the case where $M \gg k$:

$$\mathbb{T}(\mathtt{A}^\star) = \sum_{t=0}^{M} \mathtt{Pr}\left[\text{not found by time } t\right] = \sum_{t=0}^{M} \prod_{i=0}^{t-1} \left(1 - \frac{1}{M-i}\right)^k = \sum_{t=0}^{M} \prod_{i=0}^{t-1} \left(\frac{M-i-1}{M-i}\right)^k = \sum_{t=0}^{M} \left(\frac{M-t}{M}\right)^k$$

$$= \frac{1}{M^k} \sum_{i=0}^{M} i^k \approx \frac{1}{M^k} \frac{M^{k+1}}{k+1} = \frac{M}{k+1}$$

Note that with coordination, the expected running time would be about $M/2k$, so we lose about a factor of 2 by non-coordination as opposed to 4 in the case of Algorithm $\mathtt{A}_{universal}$. This algorithm is memory intensive, yet if we choose to simplify and just choose uniformly at random a box from all boxes at each step, we get that the running time is practically the same for large $M$:

$$\sum_{t=0}^{\infty} \left(1 - \frac{1}{M}\right)^{kt} = \frac{1}{1 - \left(1 - \frac{1}{M}\right)^k} \approx \frac{M}{k}$$

## 4 Pareto Distributions

$\mathtt{A}^\star$ is optimal, but it is a complex algorithm. For a large family of Pareto distributions we present a simplified algorithm that approximates the performance of $\mathtt{A}^\star$ well. Let $r_{b,M}$ be the Pareto distribution with parameter $b > 0$ on $M$ boxes. Denote $b(x) = 1/x^b$, and then $r_{b,M}(x) = I/b(x)$, where $I = 1/\sum_{x=1}^{M} b(x)$ is the normalization factor. Note that the function $b(\cdot)$ will be important on its own right. We will especially be interested in the case[3] where $b < 1$, as when $M$ grows, the fraction of the weight any specific box has goes to 0. For $b > 1$ that is not true, and so we are left with too little leeway for simplifying $\mathtt{A}^\star$.

In Algorithm $\mathtt{A}_{pareto}$, each agent, at its $t$-th step, chooses uniformly from one of the boxes it did not check yet in $\{1, \ldots, \min(M, \lfloor t/\sigma \rfloor)\}$, where $\sigma = b/(b + k - 1)$. While $\mathtt{A}_{pareto}$ is not optimal, asymptotically it is. All missing proofs of the section appear in [20]. In what follows, $o(1)$ means an expression that tends to 0 as $M$ goes to infinity.

[3] In fact, our lower bound result also hold for $b = 1$, but our upper bound proof does not work for this case. However, we strongly believe the theorem to be true for $b = 1$ as well.

### 4.1 Lower Bound

The lower bound part of Theorem 4 is proved for all non-coordinating algorithms. For that, instead of the set of functions in $\mathcal{V}$, we consider a more general class of functions and so lower bound the original question. For a measurable set $X$ denote:

$$\mathcal{F}(X) = \{N : X \times [0,\infty] \to [0,1] \mid N(\cdot,t) \text{is measurable for every fixed } t\}$$

For an $N \in \mathcal{F}(X)$, we say that $N$ satisfies the *column requirements* if for all $t$: $C_N(t) = \int_X 1 - N(x,t)\,\mathrm{d}x \leq t$. Such a function is called *valid*, and $\mathcal{V}(X)$ is the set of all valid functions. Given an integer $k \geq 2$ and some measurable function $p : X \to [0,\infty)$, define:

$$\mathbb{U}_{p,k}(N) = \int_0^\infty \int_X p(x)N(x,t)^k \,\mathrm{d}x\,\mathrm{d}t$$

This is a sort of equivalent of the $\mathbb{T}$ of algorithms, but is "unnormalized", as $p$ is not necessarily a distribution. The following claim shows a connection between algorithms and functions:

**Proposition 1.** *For every distribution $p$ on $\{1,2,\ldots,M\}$ and algorithm $A$ on the $M$ boxes, there is a function $N \in \mathcal{V}([1,M+1])$ such that $\mathbb{U}_{p',k}(N) \leq \mathbb{T}_{p,k}(A)$, where $p' : [1,M+1] \to [0,\infty)$ is any non-increasing measurable function that agrees with $p$.*

It is proved quite directly by taking $N(x,t) = A(\lfloor x \rfloor, \lfloor t \rfloor)$. This shows that lower bounding the "running time" of functions in $\mathcal{V}([1,M+1])$ will lower bound the running time of algorithms on $M$ boxes. Next, fix some $0 < b < 1$, and so the function $b(x) = 1/x^b$.

**Observation 7.** *Let $X$ be a finite interval of $\mathbb{R}^+$. Among all functions of $N \in \mathcal{V}(X)$ there is one that minimizes $\mathbb{U}_{b,k}(N)$. Denote it $\mathtt{OPT}_{b,X}$.*

The proof of this observation uses the full power of Lemma 4 by finding the optimal function of $x$ for each specific $t$, in a very similar way to the optimality proof of $\mathtt{A}^\star$. Next, we introduce the important tool of *zooming*, which is used a couple of times in what follows.

**Definition 1.** *Given some $N \in \mathcal{F}(X)$ and $u,v > 0$, define the zooming of $N$ by $(u,v)$ as: $N_{\overrightarrow{u,v}}(x,t) = N(x/u, t/v)$, where $N_{\overrightarrow{u,v}}(x,t) \in \mathcal{F}(uX)$.*

The intuitive meaning of it is that the algorithm is expanded to work on a domain of size $u$ times the original one, and slowed down by a factor of $v$. What happens to the column requirement integrals and to the time?

**Lemma 6.** *For $N \in \mathcal{F}(X)$ and $u,v > 0$, $\mathbb{U}(N_{\overrightarrow{u,v}}) = u^{1-b}v\mathbb{U}(N)$, and for all $t$, $C_{N_{\overrightarrow{u,v}}}(t) = uC_N\left(\frac{t}{v}\right)$.*

This Lemma reduces our question to the running time of a specific set of optimal functions:

**Lemma 7.** *For any Algorithm $A$ that works on $M$ boxes, denoting $r = r_{b,M}$, and $\epsilon = 1/(M+1)$:*

$$\frac{\mathbb{T}_r(A)}{\mathbb{T}_r(\mathtt{A}_{cord})} \geq (1 - o(1)) \cdot k(2-b) \cdot \mathbb{U}_b(\mathtt{OPT}_{[\epsilon,1]})$$

To use Lemma 7 one should figure out who is $\mathtt{OPT}_{[\epsilon,1]}$. This is possible using Lemma 4, but the equations that calculate $\alpha(t)$ are differential and it is not clear how to solve them. However, assuming $M$ is large, we can trick our way out of this via a clever use of zooming, and so reduce the problem to calculating $\mathtt{OPT}_{(0,1]}$ which is much simpler. Denote $\mathtt{OPT} = \mathtt{OPT}_{(0,1]}$. Then:

**Lemma 8.** $\lim_{\epsilon \to 0} \left(\mathbb{U}(\mathtt{OPT}_{[\epsilon,1]})/\mathbb{U}(\mathtt{OPT})\right) = 1$

All that is left to do, is figure out $\mathtt{OPT}$ and calculate its running time:

**Lemma 9.** *Denote $\sigma = b/(b+k-1)$. Then, $\mathbb{U}(\mathtt{OPT}) = \frac{\sigma(2-\sigma)}{2-b} + \frac{(1-\sigma)^2}{k+1}$.*

Finally, we can prove the lower bound and optimality part of Theorem 4. By Lemmas 7, 8 and 9, for every algorithm $A$:

$$\lim_{M\to\infty} \frac{\mathbb{T}(A)}{\mathbb{T}(\mathtt{A}_{cord})} \geq k(2-b) \lim_{M\to\infty} \mathbb{U}(\mathtt{OPT}_{[1/(M+1),1]}) =$$
$$k(2-b)\mathbb{U}(\mathtt{OPT}) = k\sigma(2-\sigma) + \frac{k(2-b)(1-\sigma)^2}{k+1}$$

### 4.2 Upper Bound

Below we describe the high level structure of the proof of the upper bound part of Theorem 4. A simple analysis of $\mathtt{A}_{cord}$ and gives:

**Lemma 10.**

$$\frac{\mathbb{T}(\mathtt{A}_{pareto})}{\mathbb{T}(\mathtt{A}_{cord})} \leq \frac{k(2-b)}{M^{2-b}} \sum_t \sum_{x=1}^{M} \frac{1}{x^b} \mathtt{A}_{pareto}(x,t)^k$$

Since $\mathtt{A}_{pareto}$ chooses uniformly from a set of unopened boxes at each stage, by Observation 5, when $x$ is in this set then:

$$\mathtt{A}_{pareto}(x,t) = \mathtt{A}_{pareto}(x,t-1) \cdot \left(1 - \frac{1}{|\text{interval chosen from}| - (t-1)}\right)$$

Applying generously and then using Lemma 2, one gets:

**Proposition 2.**

$$\mathtt{A}_{pareto}(x,t) \leq (1+o(1)) \cdot \begin{cases} 1 & t < \lceil \sigma x \rceil \\ \left(\frac{\lceil \sigma x \rceil}{t}\right)^{\frac{b}{k-1}} & \lceil \sigma x \rceil \leq t < \lceil \sigma M \rceil \\ \frac{1}{1-\sigma}\left(1 - \frac{t}{M}\right)\left(\frac{\lceil \sigma x \rceil}{\sigma M}\right)^{\frac{b}{k-1}} & \lceil \sigma M \rceil \leq t < M \\ 0 & t \geq M \end{cases}$$

The point of this is that completely ignoring the rounding up operations, this is exactly $\texttt{OPT}(x/M, t/M)$. Indeed, using very careful needlework math to get rid of these roundings, we show what would otherwise be a simple claim:

**Proposition 3.**

$$\frac{1}{M^{2-b}} \sum_{t=0}^{M} \sum_{x=1}^{M} \frac{1}{x^b} \mathtt{A}_{pareto}(x,t)^k \leq (1+o(1))\mathbb{U}(\texttt{OPT})$$

Plugging this into Lemma 10 gives:

$$\frac{\mathbb{T}(\mathtt{A}_{pareto})}{\mathbb{T}(\mathtt{A}_{cord})} \leq (1+o(1))k(2-b)\mathbb{U}(\texttt{OPT})$$

Lemma 9 gives the value of $\mathbb{U}(\texttt{OPT})$, and concludes the upper bound proof of Theorem 4 in exactly the same fashion as the end of the lower bound proof of this theorem.

## References


1. Alon, N., Avin, C., Koucky, M., Kozma, G., Lotker, Z., Tuttle, M.R.: Many random walks are faster than one. In: Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures, SPAA 2008, New York, NY, USA, pp. 119–128. ACM (2008)
2. Alpern, S., Fokkink, R., Gasieniec, L.A., Lindelauf, R., Subrahmanian, V.S. (eds.): Search Theory: A Game Theoretic Perspective. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-6825-7
3. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. International Series in Operations Research & Management Science. Springer, Heidelberg (2003). https://doi.org/10.1007/b100809
4. Baezayates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **106**(2), 234–252 (1993)
5. Beck, A.: On the linear search problem. Israel J. Math. **2**(4), 221–228 (1964)
6. Blackwell, D.: Notes on Dynamic Programming. Unpublished notes, University of California, Berkeley (1962)
7. Cooper, C., Frieze, A.M., Radzik, T.: Multiple random walks in random regular graphs. SIAM J. Discrete Math. **23**(4), 1738–1761 (2009)
8. Czyzowicz, J., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J.: Search on a line with faulty robots. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, New York, NY, USA, pp. 405–414. ACM (2016)
9. Czyzowicz, J., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J., Shende, S.M.: Linear search with terrain-dependent speeds. CoRR, abs/1701.03047 (2017)
10. Das, S.: Mobile agents in distributed computing: network exploration. Bull. Eur. Assoc. Theor. Comput. Sci. (EATCS) **109**, 54–69 (2013)
11. David, A., Shmuel, Z.: Optimal sequential search: a Bayesian approach. Ann. Stat. **13**(3), 1213–1221 (1985)

12. Efremenko, K., Reingold, O.: How well do random walks parallelize? In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX/RANDOM -2009. LNCS, vol. 5687, pp. 476–489. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03685-9_36
13. Elsässer, R., Sauerwald, T.: Tight bounds for the cover time of multiple random walks. Theor. Comput. Sci. **412**(24), 2623–2641 (2011)
14. Feinerman, O., Korman, A.: Memory lower bounds for randomized collaborative search and implications for biology. In: Aguilera, M.K. (ed.) DISC 2012. LNCS, vol. 7611, pp. 61–75. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33651-5_5
15. Feinerman, O., Korman, A., Lotker, Z., Sereni, J.-S.: Collaborative search on the plane without communication. In: ACM Symposium on Principles of Distributed Computing, PODC 2012, Funchal, Madeira, Portugal, 16–18 July 2012, pp. 77–86 (2012)
16. Flocchini, P., Prencipe, G., Santoro, N.: Distributed Computing by Oblivious Mobile Robots. Morgan & Claypool Publishers, San Rafael (2012)
17. Fraigniaud, P., Korman, A., Rodeh, Y.: Parallel exhaustive search without coordination. In: Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, 18–21 June 2016, pp. 312–323 (2016)
18. BOINC. https://boinc.berkeley.edu/
19. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1993, Philadelphia, PA, USA, pp. 441–447. Society for Industrial and Applied Mathematics (1993)
20. Korman, A., Rodeh, Y.: Parallel search with no coordination. CoRR, abs/1705.05704 (2017)
21. Chew, M.C.: A sequential search procedure. Ann. Math. Stat. **38**(2), 494–502 (1967)
22. Prencipe, G.: Autonomous mobile robots: a distributed computing perspective. In: Flocchini, P., Gao, J., Kranakis, E., Meyer auf der Heide, F. (eds.) ALGOSENSORS 2013. LNCS, vol. 8243, pp. 6–21. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-45346-5_2
23. Stone, L.D.: Theory of Optimal Search, 2nd edn. Topics in Operations Research Series. Military Applications Section (1989)

# Monitoring of Domain-Related Problems in Distributed Data Streams

Pascal Bemmann, Felix Biermeier, Jan Bürmann, Arne Kemper, Till Knollmann, Steffen Knorr, Nils Kothe, Alexander Mäcker, Manuel Malatyali(✉), Friedhelm Meyer auf der Heide, Sören Riechers, Johannes Schaefer, and Jannik Sundermeier

Computer Science Department, Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany
{pbemmann,felixbm,jbuerman,kempera,tillk,stknorr,nkothe,amaecker, malatya,fmadh,soerenri,jschaef,janniksu}@mail.uni-paderborn.de

**Abstract.** Consider a network in which $n$ distributed nodes are connected to a single server. Each node continuously observes a data stream consisting of one value per discrete time step. The server has to continuously monitor a given parameter defined over all information available at the distributed nodes. That is, in any time step $t$, it has to compute an output based on all values currently observed across all streams. To do so, nodes can send messages to the server and the server can broadcast messages to the nodes. The objective is the minimisation of communication while allowing the server to compute the desired output.

We consider monitoring problems related to the domain $D_t$ defined to be the set of values observed by at least one node at time $t$. We provide randomised algorithms for monitoring $D_t$, (approximations of) the size $|D_t|$ and the frequencies of all members of $D_t$. Besides worst-case bounds, we also obtain improved results when inputs are parameterised according to the similarity of observations between consecutive time steps. This parameterisation allows to exclude inputs with rapid and heavy changes, which usually lead to the worst-case bounds but might be rather artificial in certain scenarios.

## 1 Introduction

Consider a system consisting of a huge amount of nodes such as a distributed sensor network. Each node continuously observes its environment and measures information such as temperature, pollution or similar parameters. Given such a system, we are interested in aggregating information and continuously monitoring properties describing the current status of the system at a central server.

This work was partially supported by the German Research Foundation (DFG) within the Priority Program "Algorithms for Big Data" (SPP 1736) and by the Federal Ministry of Education and Research (BMBF) as part of the project "Resilience by Spontaneous Volunteers Networks for Coping with Emergencies and Disaster" (RESIBES), (grant nos. 13N13955 to 13N13957).


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 212–226, 2017.
https://doi.org/10.1007/978-3-319-72050-0_13

To keep the server's information up to date, the server and the nodes can communicate with each other. In sensor networks, however, the amount of such communication is particularly crucial, as communication translates to energy consumption, which determines the overall lifetime of the network due to limited battery capacities. Therefore, algorithms aim at minimizing the communication required for monitoring the respective parameter at the server.

One very basic parameter is the domain of the system defined to be the values currently observed across all nodes. We consider different notions related to the domain and propose algorithms for monitoring the domain itself, (approximations of) its size and (approximations of) the frequencies of values comprising the domain, respectively. Each of these parameters can provide useful information, e.g. the information about the (approximated) frequency of each value allows to approximate very precisely the histogram of the observed values, and this allows to determine (approximations of) several functions of the input, e.g. heavy hitters, quantiles, top-$k$, frequency moments or threshold problems.

### 1.1 Model and Problems

We consider the continuous distributed monitoring setting, introduced by Cormode et al. [1], in which there are $n$ distributed nodes, each uniquely identified by an identifier (ID) from the set $\{1, \ldots, n\}$, connected to a single server. Each node observes a stream of values over time and at any discrete time step $t$ node $i$ observes one value $v_i^t \in \{1, \ldots, \Delta\}$. The server is asked to, at any point $t$ in time, compute an output $f(t)$ which depends on the values $v_i^{t'}$ (for $t' \leq t$, and $i = 1, \ldots, n$) observed across all distributed streams up to the current time step $t$. The exact definition of $f(\cdot)$ depends on the concrete problems under consideration, which are defined in the section below. For the solution of these problems, we are usually interested in approximation algorithms. An $\varepsilon$-approximation of $f(t)$ is an output $\tilde{f}(t)$ of the server such that $(1-\varepsilon)f(t) \leq \tilde{f}(t) \leq (1+\varepsilon)f(t)$. We call an algorithm that, for each time step, provides an $\varepsilon$-approximation with probability at least $1-\delta$, an $(\varepsilon, \delta)$-approximation algorithm. To be able to compute the output, the nodes and the server can communicate with each other by exchanging single cast messages or by broadcast messages sent by the server and received by all nodes. Both types of communication are instantaneous and have unit cost per message. That is, sending a single message to one specific node incurs cost of one and so does one broadcast message. Each message has a size of $O(\log \Delta + \log n + \log\log \frac{1}{\delta})$ bits and will usually, besides a constant number of control bits, consist of a value from $\{1, \ldots, \Delta\}$, a node ID and an identifier to distinguish between messages of different instances of an algorithm applied in parallel (as done when using standard probability amplification techniques). Having a broadcast channel is an extension to [1], which was originally proposed in [2] and afterwards applied in [7,8]. For ease of presentation, we assume that not only the server can send broadcast messages, but also the nodes. This changes the communication cost only by a factor of at most two. Between any two time steps we allow a communication protocol to take place, which may use polylogarithmic $\mathcal{O}(\log^c n)$ rounds, for some constant $c$. The optimisation goal is

the minimisation of the communication cost, given by the number of exchanged messages, required to monitor the considered problem.

**Monitoring of Domain-Related Functions.** In this paper, we consider the monitoring of different problems related to the *domain* of the network. The domain at time $t$ is defined as $D_t := \{v \in \{1, \ldots, \Delta\} \mid \exists i \text{ with } v_i^t = v\}$, the set of values observed by at least one node at time $t$. We study the following three problems related to the domain:

- **Domain Monitoring.** At any point in time, the server needs to know the domain of the system as well as a *representative* node for each value of the domain. Formally, monitor $D_t = \{v_1, \ldots, v_{|D_t|}\} \subseteq \{1, \ldots, \Delta\}$, at any point $t$ in time. Also, maintain a sequence $R_t = (j_1, \ldots, j_\Delta)$ of nodes such that for all observed values $v \in D_t$ a representative $i$ is determined with $j_v = i$ and $v_i^t = v$. For each value $v \notin D_t$ which is not observed, no representative is given and $j_v = \text{nil}$.
- **Frequency Monitoring.** For each $v \in D_t$ monitor the frequency $|N_t^v|$ of nodes in $N_t^v := \{i \in \{1, \ldots, n\} \mid v_i^t = v\}$ that observed $v$ at $t$, i.e. the number of nodes currently observing $v$.
- **Count Distinct Monitoring.** Monitor $|D_t|$, i.e. the number of distinct values observed at time t.

We provide an exact algorithm for the Domain Monitoring Problem and $(\varepsilon, \delta)$-approximations for the Frequency and Count Distinct Monitoring Problem.

### 1.2 Our Contribution

For the Domain Monitoring Problem, an algorithm which uses $\Theta(\sum_{t \in T} |D_t|)$ messages in expectation for $T$ time steps is given in Sect. 2. This is asymptotically optimal in the worst-case in which $D_t \cap D_{t+1} = \emptyset$ holds for all $t \in T$. We also provide an algorithm and an analysis based on the minimum possible number $R^*$ of changes of representatives for a given input. It exploits situations where $D_t \cap D_{t+1} \neq \emptyset$ and uses $\mathcal{O}(\log n \cdot R^*)$ messages in expectation.

For an $(\varepsilon,\delta)$-approximation of the Frequency Monitoring Problem for $T$ time steps, we first provide an algorithm using $\Theta(\sum_{t \in T} |D_t| \frac{1}{\varepsilon^2} \log \frac{|D_t|}{\delta})$ messages in expectation in Sect. 3. We then improve this bound for instances in which observations between consecutive steps have a certain similarity. That is, for inputs fulfilling the property that for all $v \in \{1, \ldots, \Delta\}$ and some $\sigma \leq 1/2$, the number of nodes observing $v$ does not change by a factor larger than $\sigma$ between consecutive time steps, we provide an algorithm that uses an expected amount of $\mathcal{O}(|D_1|(\max(\delta, \sigma)T + 1) \frac{1}{\varepsilon^2} \log \frac{|D_1|}{\delta})$ messages. In Sect. 4, we provide an algorithm using $\Theta(T \cdot \frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ messages in expectation for the Count Distinct Monitoring Problem for $T$ time steps. For instances which exhibit a certain similarity an algorithm is presented which monitors the problem using $\Theta\left((1 + T \cdot \max\{2\sigma, \delta\}) \frac{\log(n) \cdot R^*}{d_{min} \cdot \varepsilon^2} \log \frac{1}{\delta}\right)$ messages in expectation, where $d_{min}$ denotes the minimal size of $|D_t|$ within $T$ time steps.

### 1.3 Related Work

The basis of the model considered in this paper is the *continuous monitoring model* as introduced by Cormode et al. [1]. In this model, there is a set of $n$ distributed nodes each observing a stream given by a multiset of items in each time step. The nodes can communicate with a central server, which in turn has the task to continuously, at any time $t$, compute a function $f$ defined over all data observed across all streams up to time $t$. The goal is to design protocols aiming at the minimisation of the number of bits communicated between the nodes and the server. In [1], the monitoring of several functions is studied in their (approximate) threshold variants, in which the server has to output 1 if $f \geq \tau$ and 0 if $f \leq (1-\varepsilon)\tau$, for given $\tau$ and $\varepsilon$. Precisely, algorithms for the frequency moments $F_p = \sum_i m_i^p$ where $m_i$ denotes the frequency of item $i$ for $p = 0, 1, 2$ are given. $F_1$ represents the simple sum of all items received so far and $F_0$ the number of distinct items received so far. Since the introduction of the model, monitoring of several functions has been studied such as the monitoring of frequencies and ranks by Huang et al. [5]. The frequency of an item $i$ is defined to be the number of occurrences of $i$ across all streams up to the current time. The rank of an item $i$ is the number of items smaller than $i$ observed in the streams. Frequency moments for any $p > 2$ are considered by Woodruff and Zhang in [9]. A variant of the Count Distinct Monitoring Problem is considered by Gibbons and Tirthapura in [4]. The authors study a model in which each of two nodes receives a stream of items and at the end of the streams a server is asked to compute $F_0$ based on both streams. A main technical ingredient is the use of so called public coins, which, once initialized at the nodes, provide a way to let different nodes observe identical outcomes of random experiments without further communication. We will adopt this technique in Sect. 4. Note that the previously mentioned problems are all defined over the items *received so far*, which is in contrast to the definition of monitoring problems which we are going to consider and which are all defined only based on the *current time step*. This fact has the implication that in our problems the monitored functions are no longer monotone, which makes its monitoring more complicated.

Concerning monitoring problems in which the function tracked by the server only depends on the current time step, there is also some previous work to mention. In [6], Lam et al. study a setting in which the server needs to know, at any time, the order type of the values currently observed. That is, the server needs to know which node observes the largest value, second largest value and so on at time $t$. In [10], Yi and Zhang consider a system only consisting of one node connected to the server. The node continuously observes a $d$-dimensional vector of integers from $\{1, \ldots, \Delta\}$. The goal is to keep the server informed about this vector up to some additive error per component. In [3], Davis et al. consider the following resource allocation problem: $n$ nodes observe streams of required shares of a given resource. The server has to assign, to each node, in each time step, a share of the resource that is as least as large as the required share. The objective is then given by the minimization of communication necessary for adapting the assignment of the resource over time.

## 2 The Domain Monitoring Problem

We start by presenting an algorithm to solve the Domain Monitoring Problem for a single time step. We analyse the communication cost using standard worst-case analysis and show tight bounds. By applying the algorithm for each time step, we then obtain tight bounds for monitoring the domain for any $T$ time steps. The basic idea of the protocol as given in Algorithm 1 is quite simple: Applied at a time $t$ with a value $v \in \{1, \ldots, \Delta\}$, the server gets informed whether $v \in D_t$ holds or not. To do so, each node $i$ with $v_i^t = v$ essentially draws a value from a geometric distribution and then those nodes having drawn the largest such value send broadcast messages. By this, one can show that in expectation only a constant number of messages is sent.

Furthermore, if applied with $v = nil$, the server can decide whether $v' \in D_t$ for all $v' \in \{1, \ldots, \Delta\}$ at once with $\Theta(|D_t|)$ messages in expectation. To this end, for *each* $v' \in \{1, \ldots, \Delta\}$ independently, the nodes $i$ with $v_i^t = v'$ drawing the largest value from the geometric distribution send broadcast messages. In the presentation of Algorithm 1, we assume that $v_i^t = v$ is always true if $v = nil$. Also, in order to apply it to a subset of nodes, we assume that each node maintains a value $status_i \in \{0, 1\}$ and only nodes $i$ take part in the protocol for which $status_i = status$ holds.

**Algorithm 1.** ConstantResponse$(v, status)$ *[for fixed time t]*

1. Each node $i$ for which $status_i = status$ and $\big(v \neq nil \Rightarrow v_i^t = v\big)$ hold, draws a value $\hat{h}_i$ from a geometric distribution with success probability $p := 1/2$.
2. Let $h_i = \min\{\log n, \hat{h}_i\}$.
3. Node $i$ broadcasts its value in round $\log n - h_i$ unless a node $i'$ with $v_i^t = v_{i'}^t$ has broadcasted before.

We have the following lemma, which bounds the expected communication cost of Algorithm 1 and has already appeared in a similar way in [8] (Lemma III.1).

**Lemma 1.** *Applied for a fixed time $t$,* ConstantResponse$(v, 1)$ *uses $\Theta(1)$ messages in expectation if $v \neq nil$ and $\Theta(|D_t|)$ otherwise.*

In order to solve the domain monitoring problem for $T$ time steps, the server proceeds as follows: In each step $t$ the server calls ConstantResponse$(nil, 1)$ to identify all values belonging to $D_t$ as well as a valid sequence $R_t$. By the previous lemma we then have an overall communication cost of $\Theta(|D_t|)$ for each time step $t$. For monitoring $T$ time steps, the cost is $\Theta(\sum_{t \in T} |D_t|)$. This is asymptotically optimal in the worst-case since on instances where $D_t \cap D_{t+1} = \emptyset$ for all $t$, any algorithm has cost $\Omega(\sum_{t \in T} |D_t|)$.

**Theorem 1.** *Using* ConstantResponse$(v, 1)$*, the Domain Monitoring Problem for $T$ time steps can be solved using $\Theta(\sum_{t \in T} |D_t|)$ messages in expectation.*

### A Parameterised Analysis

Despite the optimality of the result, the strategy of computing a new solution from scratch in each time step seems unwise and the analysis does not seem to capture the essence of the problem properly. It often might be the case that there are some similarities between values observed in consecutive time steps and particularly, that $D_t \cap D_{t+1} \neq \emptyset$. In this case, there might be the chance to keep a representative for several consecutive time steps, which should be exploited. Due to these observations we next define a parameter describing this behavior and provide a parameterised analysis. To this end, we consider the number of component-wise differences in the sequences of nodes $R_{t-1}$ and $R_t$ and call this difference the *number of changes of representatives* in time step $t$. Let $R^*$ denote the minimum possible number of changes of representatives (over all considered time steps $T$). The formal description of our algorithm is given in Algorithm 2. Roughly speaking, the algorithm defines, for each value $v$, phases, where a phase is defined as a maximal time interval during which there exists one node observing value $v$ throughout the entire interval. Whenever a node being a representative for $v$ changes its observation, it informs the server so that a new representative can be chosen (from those observing $v$ throughout the entire phase, which is indicated by $status_i = 1$). If no new representative is found this way, the server tries to find a new representative among those observing $v$ and for which $status_i = 0$ and ends the current phase. Additionally, if a node observes a value $v$ at time $t$ for which $v \notin D_t$, a new representative is determined among these nodes. Note that this requires each node to store $D_t$ at any time $t$ and hence a storage of $\mathcal{O}(\Delta)$.

**Theorem 2.** DomainMonitoring *as described in Algorithm 2 solves the Domain Monitoring Problem using* $\mathcal{O}(\log n \cdot R^*)$ *messages in expectation, where* $R^*$ *denotes the minimum possible number of changes of representatives.*

*Proof.* We consider each value $v \in \bigcup_t D_t$ separately. Let $N_{t_1,t_2} := \{i \mid v_i^t = v \, \forall t_1 \leq t \leq t_2\}$ denote the set of nodes that observe the value $v$ at each point in time $t$ with $t_1 \leq t \leq t_2$. Consider a fixed phase for $v$ and let $t_1$ and $t_2$ be the points in time where the phase starts and ends, respectively. A phase only ends in Step 3), hence there was no response from ConstantResponse$(v, 1)$, which implies $N^v_{t_1,t_2} = \emptyset$. Thus, to each phase for $v$ we can associate a cost of at least one to $R^*$ and this holds for each $v \in \bigcup_t D_t$. Therefore, $R^*$ is at least the overall number of phases of all values.

Next we analyze the expected cost of Algorithm 2 during the considered phase for $v$. Let w.l.o.g. $N_{t_1} := N_{t_1,t_1} = \{1, 2, \ldots, k\}$. With respect to the fixed phase, only nodes in $N_{t_1}$ can communicate and the communication is bounded by the number of changes of the representative for $v$ during the phase. Let $t'_i$ be the first time after $t_1$ at which node $i$ does not observe $v$. Let the nodes be sorted such that $i < j$ implies $t'_i \geq t'_j$. Let $a_1, \ldots, a_m$ be the nodes Algorithm 2 chooses as representatives in the considered phase. We want to show that $\mathbb{E}[m] = \mathcal{O}(\log k)$. To this end, partition the set of time steps $t'_i$ into groups $G_i$. Intuitively, $G_i$ represents the time steps in which the nodes continuously observe value $v$

**Algorithm 2.** DomainMonitoring

**(Node $i$)**

1. Define $status_i := 1$.
2. If at some time $t$, $v_i^t \neq v_i^{t-1}$, then
   2.1 If $v_i^t \notin D_{t-1}$, set $status_i = 0$ and apply ConstantResponse$(v_i^t, 0)$.
   2.2 If $v_i^t \in D_{t-1}$, set $status_i = 0$. Additionally inform server in case $i \in R_{t-1}$.
   2.3 If server starts a new phase for $v = v_i^t$, set $status_i = 1$.

**(Server)**

*[Initialisation]*
Call ConstantResponse$(nil, 1)$ to define $D_0$ and for each $v \in D_0$ choose a representative uniformly at random from all nodes which have sent $v$.

*[Maintaining $D_t$ and $R_t$ at time $t$]*
Start with $D_t = D_{t-1}$ and $R_t = R_{t-1}$ and apply the following rules:

- *[Current Phase, (try to) find new representative]*
  If informed by representative of a value $v \in D_{t-1}$,
  1) Call ConstantResponse$(v, 1)$.
  2) If node(s) respond(s), choose new representative among the responding sensors uniformly at random.
  3) Else call ConstantResponse$(v, 0)$. End current phase for $v$ and, if there is no response, delete $v$ from $D_t$ and the respective representative from $R_t$.
- *[If ConstantResponse$(v, 0)$ leads to received message(s), start new phase]*
  Start a new phase for value $v$ if message from an application of ConstantResponse$(v, 0)$ (by Step 3) initialised by the server or initialised in Step 2.1. by a node) is received. Add or replace respective representative in $R_t$ by choosing a node uniformly at random from those responding to ConstantResponse$(v, 0)$.

since time $t_1$ and the size of the initial set of nodes that observed $v$ is halved $i$ times. Formally, $G_i$ contains all time steps $t_{\ell_{i-1}+1}, \ldots, t_{\ell_i}$ (where $\ell_{-1} := 0$ for convenience) such that $\ell_i$ is the largest integer fulfilling $|N_{t_1, t'_{\ell_i}}| \in (k/2^{i+1}, k/2^i]$.

Let $S_i$ be the number of changes of representatives in time steps belonging to $G_i$. We have $\mathbb{E}[m] = \sum_{i=0}^{\log k} \mathbb{E}[S_i]$. Consider a fixed $S_i$. Let $\mathcal{E}_j$ be the event that the $j$-th representative chosen in time steps belonging to $G_i$ is the first one with an index in $\left\{1, \ldots, \lfloor \frac{k}{2^{i+1}} \rfloor\right\}$. Observe that as soon as this happens, the respective representative will be the last one chosen in a time step belonging to group $G_i$.

Now, since the algorithm chooses a new representative uniformly at random from the index set $\left\{1, \ldots, \lfloor \frac{k}{2^i} \rfloor\right\}$, the probability that it chooses a representative from $\left\{1, \ldots, \lfloor \frac{k}{2^{i+1}} \rfloor\right\}$ is at least $1/2$ except for the first representative of $v$, where it might be slightly smaller due to rounding errors. $\mathcal{E}_j$ occurs only if the first $j-1$ representatives were each *not* chosen from this set, i.e. $\Pr[\mathcal{E}_j] \leq \left(\frac{1}{2}\right)^{j-2}$. Hence, $\mathbb{E}[S_i] = \sum_j \mathbb{E}[S_i | \mathcal{E}_j] \cdot \Pr[\mathcal{E}_j] \leq \sum_j j \cdot (\frac{1}{2})^{j-2} = \sum_j \frac{j}{2^{j-2}} = \mathcal{O}(1)$. □

## 3 The Frequency Monitoring Problem

In this section we design and analyse an algorithm for the Frequency Monitoring Problem, i.e. to output (an approximation) of the number of nodes currently observing value $v$. We start by considering a single time step and present an algorithm which solves the subproblem to output the number of nodes that observe $v$ within a constant multiplicative error bound. Afterwards, and based on this subproblem, a simple sampling algorithm is presented which solves the Frequency Monitoring Problem for a single time step up to a given (multiplicative) error bound and with demanded error probability.

While in the previous section we used the algorithm ConstantResponse with the goal to obtain a representative for a measured value, in this section we will use the same algorithm to estimate the number of nodes that measure a certain value $v$. Observe that the expected maximal height of the geometric experiment increases with a growing number of nodes observing $v$. We exploit this fact and use it to estimate the number of nodes with value $v$, while still expecting constant communication cost only. For a given a time step $t$ and a value $v \in D_t$, we define an algorithm ConstantFactorApproximation as follows: We apply ConstantResponse$(v, 1)$ with $status_i = 1$ for all nodes $i$. If the server receives the first response in communication round $r \leq \log n$, the algorithm outputs $\tilde{n}^v_{\text{const}} = 2^r$ as the estimation for $|N^v_t|$.

We show that we compute a constant factor approximation with constant probability. Then we amplify this probability using multiple executions of the algorithm and taking the median (of the executions) as a final result.

**Lemma 2.** *The algorithm* ConstantFactorApproximation *estimates the number* $|N^v_t|$ *of nodes observing the value* $v$ *at time* $t$ *up to a factor of* 8*, i.e.* $\tilde{n}^v_{const} \in [|N^v_t|/8, |N^v_t| \cdot 8]$ *with constant probability.*

*Proof.* Let $n^v$ be the number of nodes currently observing value $v$, i.e. $n^v := |N^v_t|$. Recall that the probability for a single node to draw height $h$ is $\Pr[h_i = h] = \frac{1}{2^h}$, if $h < \log n$, and $\Pr[h_i = h] = \frac{2}{2^h}$, if $h = \log n$. Hence, $\Pr[h_i \geq h] = \frac{1}{2^{h-1}}$ for all $h \in \{1, \ldots, \log n\}$.

We estimate the probability of the algorithm to fail, by analysing the cases that $\tilde{n}^v_{const}$ is larger than $\log n^v + 3$ or smaller than $\log n^v - 3$.

We start with the first case and by applying a union bound we obtain $\Pr[\exists i : h_i > \log n^v + 3] \leq \Pr[\exists i : h_i \geq \lceil \log n^v \rceil + 3] = n^v \cdot \left(\frac{1}{2}\right)^{\lceil \log n^v \rceil + 2} \leq \frac{1}{4}$.

For the latter case we bound the probability that each node has drawn a height strictly smaller than $\log n^v - 3$ by $\Pr[\forall i : h_i < \log n^v - 3] \leq \prod_i \Pr[h_i < \lceil \log n^v \rceil - 3] = \left(1 - \frac{1}{2^{\lceil \log n^v \rceil - 4}}\right)^{n^v} \leq \left(1 - \frac{8}{n^v}\right)^{n^v} \leq \frac{1}{e^8}$.

Thus, the probability that we compute an 8-approximation is lower bounded by $\Pr\left[\frac{n^v}{8} \leq 2^{h_i} \leq 8n^v\right] = 1 - \left(\Pr[\exists i : h_i > \log n^v + 3] + \Pr[\forall i : h_i < \log n^v - 3]\right) \geq 1 - \left(\frac{1}{4} + \frac{1}{e^8}\right) > 0.7$ □

We apply an amplification technique to boost the success probability to arbitrary $1 - \delta'$ using $\Theta(\log \frac{1}{\delta'})$ parallel executions of the ConstantFactorApproximation algorithm and choose the median of the intermediate results as the final output.

**Corollary 1.** *Applying $\Theta\left(\log \frac{1}{\delta'}\right)$ independent, parallel instances of* ConstantFactorApproximation*, we obtain a constant factor approximation of $|N_t^v|$ with success probability at least $1 - \delta'$ using $\Theta\left(\log \frac{1}{\delta'}\right)$ messages in expectation.*

To obtain an $(\varepsilon, \delta)$-approximation, in Algorithm 3 we first apply the ConstantFactorApproximation algorithm to obtain a rough estimate of $|N_t^v|$. It is used to compute a probability $p$, which is broadcasted to the nodes, so that every node observing value $v$ sends a message with probability $p$. Since the ConstantFactorApproximation result $\tilde{n}^v_{\text{const}}$ in the denominator of $p$ is close to $|N_t^v|$, the number of messages sent in expectation is independent of $|N_t^v|$. The estimated number of nodes observing $v$ is then given by the number of responding nodes $\bar{n}^v$ divided by $p$, which, in expectation, results in $|N_t^v|$.

**Algorithm 3.** EpsilonFactorApprox$(v \in D_t, \varepsilon, \delta)$ *[for fixed time $t$]*

**(Node $i$)**
1. Receive $p$ from the server.
2. Send a response message with probability $p$.

**(Server)**
1. Set $\delta' := \frac{\delta}{3}$
2. Call ConstantFactorApproximation$(v, \delta')$ to obtain $\tilde{n}^v_{\text{const}}$.
3. Broadcast $p = \min\left(1, \frac{24}{\varepsilon^2 \tilde{n}^v_{\text{const}}} \cdot \ln \frac{1}{\delta'}\right)$.
4. Receive $\bar{n}^v$ messages.
5. Compute and output estimated number of nodes in $N_t^v$ as $\tilde{n}^v = \bar{n}^v / p$.

**Lemma 3.** *The algorithm* EpsilonFactorApprox *as given in Algorithm 3 provides an ($\varepsilon$,$\delta$)-approximation of $|N_t^v|$.*

*Proof.* The algorithm obtains a constant factor approximation $\tilde{n}^v_{\text{const}}$ with probability $1 - \delta'$. The expected number of messages is $\mathbb{E}[\bar{n}^v] = n^v \cdot p$.

We start by estimating the conditional probability that more than $(1+\varepsilon)\, n^v p$ responses are sent under the condition that $\tilde{n}^v_{\text{const}} \leq 8n^v$ and $p < 1$. In this case we have $p = \frac{24}{\varepsilon^2 \tilde{n}^v_{\text{const}}} \cdot \ln \frac{1}{\delta'} \geq \frac{3}{\varepsilon^2 n^v} \cdot \ln \frac{1}{\delta'}$, hence using a Chernoff bound it follows $p_1 := \Pr[\bar{n}^v \geq (1+\varepsilon) n^v p \,|\tilde{n}^v_{\text{const}} \leq 8n^v \wedge p < 1] \leq e^{-\frac{\varepsilon^2}{3} n^v \cdot \frac{3}{\varepsilon^2 n^v} \cdot \ln \frac{1}{\delta'}} = \delta'$. Likewise the probability that less than $(1-\varepsilon)\, n^v p$ messages are sent under the condition that $\tilde{n}^v_{\text{const}} \leq 8n^v$ and $p < 1$ is $p_2 := \Pr[\bar{n}^v \leq (1-\varepsilon) n^v p \,|\tilde{n}^v_{\text{const}} \leq 8n^v \wedge p < 1] \leq e^{-\frac{\varepsilon^2}{2} n^v \cdot \frac{3}{\varepsilon^2 n^v} \cdot \ln \frac{1}{\delta'}} \leq e^{-\frac{3}{2} \ln \frac{1}{\delta'}} < \delta'$. Next consider the case that $\tilde{n}^v_{\text{const}} > 8n^v$ and $p < 1$ holds. Using $\Pr[\tilde{n}^v_{\text{const}} > 8n^v] \leq \Pr\left[\tilde{n}^v_{\text{const}} > 8n^v \vee \tilde{n}^v_{\text{const}} < \frac{n^v}{8}\right] \leq \delta'$ and $p_i \cdot \Pr[\tilde{n}^v_{\text{const}} \leq 8n^v] \leq p_i$ for $i \in \{1, 2\}$, $\Pr[(1-\varepsilon) n^v p < \bar{n}^v < (1+\varepsilon) n^v p \,|p < 1]$ $\geq 1 - (\Pr[\tilde{n}^v_{\text{const}} > 8n^v] + (p_1 + p_2)) \geq 1 - 3\delta' = 1 - \delta$. For the last case $p = 1$,

we have $\Pr\left[(1-\varepsilon)n^v p < \bar{n}^v < (1+\varepsilon)n^v p \,|p \geq 1\right] = 1$, by using $\bar{n}^v = n^v$. Now, $\Pr\left[(1-\varepsilon)n^v p < \bar{n}^v < (1+\varepsilon)n^v p\right] \geq 1-\delta$ directly follows. □

**Lemma 4.** *Algorithm* EpsilonFactorApprox *as given in Algorithm 3 uses* $\Theta(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$ *messages in expectation.*

**Theorem 3.** *There exists an algorithm that provides an* $(\varepsilon,\delta)$*-approximation for the Frequency Monitoring Problem for* $T$ *time steps with an expected number of* $\Theta\left(\sum_{t\in T}|D_t|\frac{1}{\varepsilon^2}\log\frac{|D_t|}{\delta}\right)$ *messages.*

*Proof.* In every time step $t$ we first identify $D_t$ by applying ConstantResponse using $\Theta(|D_t|)$ messages in expectation. On every value $v \in D_t$ we then perform algorithm EpsilonFactorApprox$(v,\varepsilon,\frac{\delta}{|D_t|})$, resulting in an amount of $\Theta\left(|D_t|\frac{1}{\varepsilon^2}\log\frac{|D_t|}{\delta}\right)$ messages in expectation for a single time step, while achieving a probability (using a union bound) of $1-\frac{|D_0|\delta}{|D_0|} = 1-\delta$ that in one time step the estimations for every $v$ are $\varepsilon$-approximations. Applied for each of the $T$ time steps, we obtain a bound as claimed. □

### A Parameterised Analysis

Applying EpsilonFactorApprox in every time step is a good solution in worst case scenarios. But if we assume that the change in the set of nodes observing a value is small in comparison to the size of the set, we can do better.

We extend the EpsilonFactorApprox such that in settings where from one time step to another only a small fraction $\sigma$ of nodes change the value they measure, the amount of communication can be reduced, while the quality guarantees remain intact. We define $\sigma$ such that

$$\forall t : \sigma \geq \frac{|N^v_{t-1} \setminus N^v_t| + |N^v_t \setminus N^v_{t-1}|}{|N^v_t|}.$$

Note that this also implies that $D_t = D_{t-1}$ holds for all time steps $t$, i.e. the set of measured values stays the same over time.

The extension is designed so that compared to EpsilonFactorApprox, also in settings with many changes the solution quality and message complexity asymptotically does not increase. The idea is the following: For a fixed value $v$, in a first time step EpsilonFactorApprox is executed (defining a probability $p$ in Step 3 of Algorithm 3). In every following time step, up to $1/\delta$ consecutive time steps, nodes that start or stop measuring a value $v$ send a message to the server with the same probability $p$, while nodes that do not observe a change in their value remain silent. In every time step $t$, the server uses the accumulated messages from the first time step and all messages from nodes that started measuring $v$ in time steps $2 \ldots t$, while subtracting all messages from nodes that stopped measuring $v$ in the time steps $2 \ldots t$. This accumulated message count is then used similarly as in EpsilonFactorApprox to estimate the total number of nodes observing $v$ in the current time step. The algorithm starts again if (a)

$1/\delta$ time steps are over, so that the probability of a good estimation remains good enough, or (b) the sum of estimated nodes to start/stop measuring value $v$ is too large. The latter is done to ensure that the message probability $p$ remains fitting to the number of nodes, ensuring a small amount of communication, while guaranteeing an $(\varepsilon, \delta)$-approximation.

Let $n_t^+, n_t^-$ be the number of nodes that start measuring $v$ in time step $t$ or that stop measuring it, respectively, i.e. $n_t^+ = |N_t^v \setminus N_{t-1}^v|, n_t^- = |N_{t-1}^v \setminus N_t^v|$, and $\bar{n}_t^+$ and $\bar{n}_t^-$ the number of them that sent a message to the server in time step $t$. In the following we call nodes contributing to $n_t^+$ and $n_t^-$ *entering* and *leaving*, respectively.

**Algorithm 4.** ContinuousEpsilonApprox($v$, $\varepsilon$, $\delta$)

**(Node $i$)**

1. If $t = 1$, take part in EpsilonFactorApprox called in Step 2 by the server.
2. If $t > 1$, broadcast a message with probability $p$ if $v_i^{t-1} = v \wedge v_i^t \neq v$ or $v_i^{t-1} \neq v \wedge v_i^t = v$.

**(Server)**

1. Set $\delta' \coloneqq \delta^2$.
2. Set $t \coloneqq 1$ and run EpsilonFactorApprox($v$, $\varepsilon/3$, $\delta$) to obtain $\bar{n}_1, p$.
3. Output $\tilde{n}_1 = \frac{\bar{n}_1}{p}$.
4. Repeat at the beginning of every new time step $t > 1$:
   4.1 Receive messages from nodes changing the observed value to obtain $\bar{n}_t^+$ and $\bar{n}_t^-$.
   4.2 Break if $t \geq 1/\delta$ or $\left(\sum_{i=1}^t \bar{n}_i^+ + \sum_{i=1}^t \bar{n}_i^-\right)/p \geq \bar{n}_1/2$.
   4.3 Output $\tilde{n}_t = \left(\bar{n}_1 + \sum_{i=1}^t \bar{n}_i^+ - \sum_{i=1}^t \bar{n}_i^-\right)/p$.
5. Go to Step 2.

**Lemma 5.** *For any $v \in D_1$, the algorithm* ContinuousEpsilonApprox *provides an $(\varepsilon,\delta)$-approximation of $|N_t^v|$.*

**Lemma 6.** *For a fixed value $v$ and $T' = \min\{\frac{1}{2\sigma}, \frac{1}{\delta}\}$, $\sigma \leq \frac{1}{2}$, time steps,* ContinuousEpsilonApprox *uses $\Theta\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta}\right)$ messages in expectation.*

**Theorem 4.** *There exists an $(\varepsilon,\delta)$-approximation algorithm for the Frequency Monitoring Problem for $T$ consecutive time steps which uses an amount of $\Theta\left(|D_1|\,(1 + T \cdot \max\{2\sigma, \delta\})\,\frac{1}{\varepsilon^2} \log \frac{|D_1|}{\delta}\right)$ messages in expectation, if $\sigma \leq 1/2$.*

*Proof.* The algorithm works by first applying ConstantResponse($nil$,1) to obtain $D_1$ and then applying ContinuousEpsilonApprox($v$, $\varepsilon$, $\delta/|D_1|$) for every $v \in D_1$. By Lemma 5 we know that in every time step and for all $v \in D_1$, the frequency of $v$ is approximated up to a factor of $\varepsilon$ with probability $1-\delta/|D_1|$. We divide the $T$ time steps into intervals of size $T' = \min\{\frac{1}{2\sigma}, \frac{1}{\delta}\}$ and perform ContinuousEpsilonApprox on each of them for every value $v \in D_1$. There are $\lceil \frac{T}{T'} \rceil \leq 1 + T \cdot \max\{2\sigma, \delta\}$ such intervals. For each of those, by Lemma 6

we need $\Theta\left(\left(1+\min\{\frac{1}{2\sigma},\frac{1}{\delta}\}\sigma\right)\cdot 1/\varepsilon^2 \log\frac{|D_1|}{\delta}\right)$ messages in expectation for each $v \in D_1$. This yields a complexity of $\Theta\left(|D_1|\,(1+T\cdot\max\{2\sigma,\delta\})\,\frac{1}{\varepsilon^2}\log\frac{|D_1|}{\delta}\right)$ due to $\min\{\frac{1}{2\sigma},\frac{1}{\delta}\}\sigma \leq \frac{1}{2\sigma}\cdot\sigma = \Theta(1)$. Using a union bound over the fail probability for every $v \in D_1$, a success probability of at least $1-\frac{|D_1|\delta}{|D_1|} = 1-\delta$ follows. □

By Theorem 3, trivially repeating the single step algorithm EpsilonFactorApprox needs $\Theta\left(T|D_1|\frac{1}{\varepsilon^2}\log\frac{|D_1|}{\delta}\right)$ messages in expectation for $T$ (because the number of nodes in $N_t^v$ for any $v \in D_1$ is at least $N_1^v/2$ in every time step of that interval). Hence, the number of messages sent when using ContinuousEpsilonApprox is reduced in the order of $\max\{2\sigma,\delta\}$.

## 4 The Count Distinct Monitoring Problem

In this section we present an ($\varepsilon$,$\delta$)-approximation algorithm for the Count Distinct Monitoring Problem. The basic approach is similar to the one presented in the previous section for monitoring the frequency of each value. That is, we first estimate $|D_t|$ up to a (small) constant factor and then use the result to define a protocol for obtaining an $(\varepsilon,\delta)$-approximation. If we could assume that, at any fixed time $t$, each value was observed by at most one node, it would be possible to solve this problem with expected communication cost of $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$ (per time step $t$ and per value $v \in D_t$) using the same approach as in the previous section. Since this assumption is generally not true, we aim at simulating such behaviour that for each value in the domain only one random experiment is applied. We apply the concept of *public coins*, which allows nodes measuring the same value to observe identical outcomes of their random experiments. To this end, nodes have access to a shared random string $R$ of fully independent and unbiased bits. This can be achieved by letting all nodes use the same pseudorandom number generator with a common starting seed, adding a constant number of messages to the bounds proven below. We assume that the server sends a new seed in each phase by only loosing at most a constant factor in the amount of communication used. However, we can drop this assumption by checking whether there are nodes that changed their value such that only in rounds in which there are changes new public randomness is needed. The formal description of the algorithm for a constant factor and an $\varepsilon$-approximation are given in Algorithms 5 and 6, respectively.

We consider the access of the public coin to behave as follows: Initialised with a seed, a node accesses the sequence of random bits $R$ bitwise, i.e. after reading the $j$'th bit, the node next accesses bit $j+1$. Observe the crucial fact that as long as each node accesses the exact same number of bits, each node observes the exact same random bits simultaneously. Algorithm 5 essentially works as follows: In a first step, each node draws a number from a geometrical distribution using the public coin. By this, all nodes observing the same value $v$ obtain the same height $h_v$. In the second step we apply the strategy as in the previous section to reduce communication if lots of nodes observe the same value: Each node $i$

draws a number $g_i$ from a geometrical distribution without using the public coin. Afterwards, all nodes with the largest height $g_i$ among those with the largest height $h_v$ broadcast their height $h_v$.

**Algorithm 5.** ConstantFactorApproximation *[for fixed time t]*

**(Node $i$, observes value $v = v_i$)**

1. Draw a random number $h_v$ as follows:
   Consider the next $\Delta \cdot \log n$ random bits $b_1, \ldots, b_{\Delta \cdot \log n}$ from $R$. Let $h$ be the maximal number of bits $b_{v \cdot \log n+1}, \ldots, b_{v \cdot \log n+1+h}$ that equal 0. Define $h_v := \min\{h, \log n\}$.
2. Let $g_i'$ be a random value drawn from a geometric distribution with success-probability $p = 1/2$ and define $g_i = \min(g_i', \log n)$ (without accessing public coins).
3. Broadcast drawn height $h_v$ in round $r = \log^2 n - (h_v - 1) \cdot \log n - g_i$ unless a node $i'$ has broadcasted before.

**(Server)**

1. Receive a broadcast message containing height $h$ in round $r$.
2. Output $\hat{d}_t = 2^h$.

Note that only (at most $n$) nodes that observe value $v$ with $h_v = max_{v'}\, h_{v'}$ may send a message in Algorithm 5. Now, all nodes observing the same value observe the same outcome of their random experiments determining $h_v$. Hence, by a similar reasoning as in Lemma 2, one execution of the algorithm uses $\mathcal{O}(1)$ messages in expectation.

Using the algorithm given in Algorithm 5 and applying the same idea as in the previous section, we obtain an $(\varepsilon, \delta)$-approximation as given in Algorithm 6: Each node tosses a coin with a success probability depending on the constant factor approximation (for which we have a result analogous to Corollary 1). Again, all nodes use the public coin so that all nodes observing the same value obtain the same outcome of this coin flip. Afterwards, those nodes which have observed a success apply the same strategy as in the previous section, that is, they draw a random value from a geometric distribution, and the nodes having the largest height send a broadcast.

Using arguments analogous to Lemmas 3 and 4 and applying EpsilonFactorApprox for $T$ time steps, we obtain the following theorem.

**Theorem 5.** *There exists an $(\varepsilon, \delta)$-approximation algorithm for the Count Distinct Monitoring Problem for $T$ time steps using $\mathcal{O}(T \cdot \frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ messages in expectation.*

### A Parameterised Analysis

In this section we consider the problem for multiple time steps and parameterise the analysis with respect to instances in which the domain does not change arbitrarily between consecutive time steps. Recall that for monitoring the frequency from a time step $t-1$ to the current time step $t$, all nodes that left and all nodes

**Algorithm 6.** EpsilonFactorApprox *[for fixed time t]*

**(Node $i$)**

1. Flip a coin with success probability $p = 2^{-q} = \frac{c \log 1/\delta}{\varepsilon^2 \tilde{d}_t}$, $q \in \mathbb{N}$ as follows: Consider the next $\Delta \cdot q$ random bits $b_1, \ldots b_{\Delta \cdot q}$. The experiment is successful if and only if all random bits $b_{v \cdot q+1}, \ldots, b_{v \cdot q+q}$ equal 0. The node deactivates (and does not take part in Steps 2. and 3.) if the experiment was not successful.
2. Draw a random value $h_i'$ from a geometric distribution and define $h_i = \min(h_i', \log n)$ (without accessing public coins).
3. Node $i$ broadcasts its value in round $\log n - h_i$ unless a node $i'$ with $v_i^t = v_{i'}^t$ has broadcasted before.

**(Server)**

1. Let $S_t$ be the set of received values.
2. Output $\tilde{d}_t := |S_t|/p$

that entered toss a coin to estimate the number of changes. However, to identify that a node observes a value which was not observed in the previous time step, the domain has to be determined exactly.

We apply the following idea instead: For each value $v \in \{1, \ldots, \Delta\}$ we flip a (public) coin. We denote the set of values with a successful coin flip as the *sample*. Afterwards, the algorithm only proceeds on the values of the sample, i.e. in cases in which a node observes a value with a successful coin flip and no node observed this value in previous time steps, this value contributes to the estimate $\tilde{d}_t^+$ at time $t$. Regarding the (sample) of nodes that leave the set of observed values, the DomainMonitoring algorithm is applied to identify which (sampled) values are not observed any longer (and thus contribute to $\tilde{d}_t^-$). Analogous to Lemma 5, we have the following lemma.

**Algorithm 7.** ContinuousEpsilonApprox$(\varepsilon, \delta)$

1. Compute $\delta' = 2\,\delta^2$
2. Broadcast a new seed value for the public coin.
3. Compute an $(\varepsilon, \delta')$-approximation $\tilde{d}_1$ of $|D_1|$ using Algorithm 6. Furthermore, obtain the success-probability $p$.
4. Repeat for each time step $t > 1$:
   4.1 Each node $i$ applies Algorithm 2 if the observed value $v_i$ is in the sample set. Let $\hat{d}_t^-$ be the number of values (in sample set) which left the domain and $\hat{d}_t^+$ the number of nodes that join the sample.
   4.2 Server computes $\tilde{d}_t = \tilde{d}_1 + \sum_{i=2}^t \hat{d}_i^+/p - \sum_{i=2}^t \hat{d}_i^-/p$.
   4.3 Break if $t = 1/\delta$ or $\left(\sum_{i=2}^t \tilde{d}_i^+ + \sum_{i=2}^t \tilde{d}_i^-\right)/p$ exceeds $\tilde{d}_1/2$.
5. Set $t = 1$ and go to Step 2.

**Lemma 7.** ContinuousEpsilonApprox *achieves an (ε,δ)-approximation of* $|D_t|$ *in any time step* $t$.

For the number of messages, we argue based on the previous section. However, in addition the DomainMonitoring algorithm is applied. Observe that the size of the domain changes by at most $n/2$, and consider the case that this number of nodes observed the same value $v$. The expected cost (where the expectation is taken w.r.t. whether $v$ is within the sample) is $\mathcal{O}(\log n \cdot R^* \cdot p) = \mathcal{O}\big(\frac{\log n \cdot R^*}{d_{min} \cdot \varepsilon^2} \log \frac{1}{\delta}\big)$, with $d_{min} := \min_t |D_t|$. Similar to Theorem 4, we then obtain this theorem.

**Theorem 6.** ContinuousEpsilonApprox *provides an* $(\varepsilon, \delta)$*-approximation for the Count Distinct Monitoring Problem for* $T$ *time steps using an amount of* $\Theta\left((1 + T \cdot \max\{2\sigma, \delta\}) \frac{\log(n) \cdot R^*}{d_{min} \cdot \varepsilon^2} \log \frac{1}{\delta}\right)$ *messages in expectation, if* $\sigma \leq 1/2$.

## References


1. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. In: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), pp. 1076–1085. SIAM (2008)
2. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. ACM Trans. Algorithms **7**(2), 21:1–21:20 (2011)
3. Davis, S., Edmonds, J., Impagliazzo, R.: Online algorithms to minimize resource reallocations and network communication. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX/RANDOM -2006. LNCS, vol. 4110, pp. 104–115. Springer, Heidelberg (2006). https://doi.org/10.1007/11830924_12
4. Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: Proceedings of the 13th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2001), pp. 281–291. ACM (2001)
5. Huang, Z., Yi, K., Zhang, Q.: Randomized algorithms for tracking distributed count, frequencies, and ranks. In: Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2012), pp. 295–306. ACM (2012)
6. Lam, T.-W., Liu, C.-M., Ting, H.-F.: Online tracking of the dominance relationship of distributed multi-dimensional data. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 178–189. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18318-8_16
7. Mäcker, A., Malatyali, M., auf der Heide, F.M.: Online Top-k-position monitoring of distributed data streams. In: Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2015), pp. 357–364. IEEE (2015)
8. Mäcker, A., Malatyali, M., auf der Heide, F.M.: On competitive algorithms for approximations of Top-k-position monitoring of distributed streams. In: Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016), pp. 700–709. IEEE (2016)
9. Woodruff, D.P., Zhang, Q.: Tight bounds for distributed functional monitoring. In: Proceedings of the 44th Symposium on Theory of Computing (STOC 2012), pp. 941–960. ACM (2012)
10. Yi, K., Zhang, Q.: Multidimensional online tracking. ACM Trans. Algorithms **8**(2), 12 (2012)

# Killing Nodes as a Countermeasure to Virus Expansion

François Bonnet[1], Quentin Bramas[2(✉)], Xavier Défago[3], and Thanh Dang Nguyen[4]

[1] Graduate School of Engineering, Osaka University, Suita, Japan
francois@cy2sec.comm.eng.osaka-u.ac.jp
[2] Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 Place Jussieu, 75005 Paris, France
quentin.bramas@lip6.fr
[3] School of Computing, Tokyo Institute of Technology, Tokyo, Japan
defago@c.titech.ac.jp
[4] University of Chicago, Chicago, USA
thanhnd@uchicago.edu

**Abstract.** The *spread of a virus* and the *containment of such spread* have been widely studied in the literature. These two problems can be abstracted as a two-players stochastic game in which one side tries to spread the infection to the entire system, while the other side aims to contain the infection to a finite area. Three parameters play a particularly important role: (1) the probability $p$ of successful infection, (2) the topology of the network, and (3) the probability $\alpha$ that a strategy message has priority over the infection.

This paper studies the effect of *killing strategies*, where a node sacrifices itself and possibly some of its neighbors, to contain the spread of a virus in an infinite grid. Our contribution is threefold: (1) We prove that the simplest killing strategy is equivalent to the problem of site percolation; (2) when killing messages have priority, we prove that there always exists a killing strategy that contains a virus, for any probability $0 \leq p < 1$; in contrast, (3) when killing message do not have priority, there is not always a successful killing strategy, and we study the virus propagation for various $0 \leq \alpha < 1$.

## 1 Introduction

Consider the propagation of a virus in a large distributed system, such as the Internet, a sensor network, or a social network. An epidemic starts with an arbitrary node being initially infected, and then continues with each newly infected node attempting to infect its neighbors. If nothing is done, all connected nodes are eventually infected, and the time it takes only depends on the infection rate (*i.e.*, the probability of success for each infection attempt), the topology of the network, and the location of the initial node.

Consider now that, upon an unsuccessful infection attempt, the targeted node has a chance to detect the attempt and react to it. One simple strategy consists


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 227–243, 2017.
https://doi.org/10.1007/978-3-319-72050-0_14

in ordering that node to inform its neighbors of the presence of the virus and then to kill itself. The informed node can have the same behavior so that the infection can no longer spread through them. The hope is that, given enough nodes are killed, they could actually isolate the infected nodes from the sane ones, thus containing the spread of the virus. The chance of this happening depend mainly on the following factors: (1) the infection rate $p$, (2) the detection probability, (3) the probability $\alpha$ that killing messages have priority over the infection, and (4) the topology of the network. In this paper we suppose that a node always detect the virus when the infection fails (the detection probability equals $1-p$).

Such systems are notoriously known to exhibit a *critical threshold*, that is, the spread is almost surely contained if the infection rate is below the threshold, but is very unlikely otherwise. This is reminiscent of another problem, known as percolation [2], where the same *phase transition* occurs.

In this paper, we look at the following question: is there always a strategy to contains the spread of a virus? We answer this question when the topology of the network is a grid and for $0 \leq \alpha \leq 1$.

In our recent work [19], we have studied this question and the impact of several containment strategies in various topologies, by relying on extensive simulations. We found that containment strategies do have an impact on the critical threshold of the system, and so does the topology. In this paper we formalize and prove some of those observations.

*Our Contributions.* The contribution of this paper is threefold. First, we show an equivalence between infection rate threshold with the simplest killing strategy and percolation threshold for the same topology. Second, we prove that, in the infinite grid and when killing message have priority, there is always a killing strategy such that the spread is contained with probability one, for any infection rate $0 \leq p < 1$. Third, we prove that, in the infinite grid and when killing message do not have priority, we give a lower bound on the threshold of the infection probability, above which the virus spreads infinitely with a positive probability.

The rest of the paper is structured as follows. Section 2 gives a brief overview of the state-of-the-art on epidemic research. Section 3 defines the model, the topology, and the killing strategies considered in the paper. Section 4 proves the equivalence between the simplest killing strategy and percolation. Finally, Sect. 5 proves that containment is always possible in an infinite grid, and Sect. 6 shows a lower bound on the threshold of the infection probability.

## 2 Related Work

**On the Spread of a Virus.** Starting from the epidemiology in human community, much research has been conducted on propagation. The first mathematical models appeared in the 18th century, but modern models were essentially developed in the middle of the 20th century (e.g., [1,12,17]). While original models did not consider geographic distributions, more recent epidemic models consider geographic topologies, such as an infinite grid [6]).

Kephart and White [10] propose a birth-death model to study the spread of computer viruses in homogeneous sparse graphs and conclude that a pandemic occurs only when the infection rate exceeds a finite threshold that depends on the connectivity of the network (phase transition). They also extend their model to allow doing a virus scan [11].

Later, many works improve the results on the birth-death model and compute new epidemic thresholds. Pastor-Satorras and Vespignani [20,21] look at the dynamics of epidemics in power-law scale free networks for which they find the critical threshold. Chakrabarti et al. [4] study an epidemic model with recovery and find that the propagation threshold is related to the eigenvalues of the adjacency matrix of the network. Lately, Van Mieghem et al. [24,25] use mean field approximation to transform from individual random infection rates into an average infection rate. Their model is called $N$-intertwined Markov chain.

In another direction, after the propagation of Code Red in 2001, many researches look for the most accurate model to reflect the spread of different kinds of viruses in the Internet. They propose different models from the *scanning worms* [23,27,30] to the *event-based* worms [28,31], where the question is to predict, as accurately as possible, the evolution of the expected number of infected entities in the network after the virus starts propagating.

**On Defense Against a Virus Propagation.** In virus defense area, there are many works studying how to contain or quarantine the virus or worms in different network environments [18,29,31]. The containment strategies can be classified into two main classes; *proactive* or *reactive*. In the first one, some nodes are initially immune to the virus and only other nodes can be infected. In the latter, all nodes are initially susceptible, but eventually any node may become immune if it detects the virus (or receive some informations from other nodes).

Several work [29,31] consider the spread viruses against proactive containment strategy, where the goal is to study the impact of the choice of the set of immune nodes on the spread of the virus.

Moore et al. [18] proposed a model for scanning worms in complete graph topology and give a comparison between two reactive strategies; (1) *blacklisting*, upon detection of an infection, a node adds the attacker into a blacklist; and (2) *filtering-content*, upon detection of a virus, a node transmits its signature to all other nodes. They assume that when a node detects an infection, the information (blacklisted IP address, or virus' signature) will be available to all other nodes after some time. They study the efficiency of both strategies when this delay varies.

In all these work, immune nodes can always detect virus attack successfully. However, with a polymorphic (such as Sality [5]) or metamorphic virus, it may not be always possible to detect the virus correctly. In this context, several detectors are introduced in [3,13,15]. We call *imperfect detection* the ability to detect a virus but not always successfully. The problem of the containment of a virus, when nodes are provided with an imperfect detector, remains open.

**Flooding and Percolation.** This containment problem is related to probabilistic broadcast (or information flooding) albeit with an opposite objective. Sasson et al. [22] and later Hu et al. [7] both study the question and show the relationship with the theory of percolation [2].

## 3 Model and Definitions

Let $G = (V, E)$ be a connected undirected infinite graph, where $V$ is the set of vertices, and $E$ the set of edges. Vertices model the nodes in the system, and edges the bidirectional communication links between a pair of nodes. Thus, edge $e_{ij} \in E$ represents a communication link between the two nodes $i, j \in V$.

The system evolves in numbered synchronous rounds, also called *timeslots*. During a round, every node can exchange messages with its direct neighbors. Time is discrete and measured according to the round number.

A node can be in one of four states:

- *infected*: the node is compromised by the virus and acts as an infectious agent.
- *killed*: the node no longer sends or receives any message.
- *susceptible*: the node is neither infected nor killed, but can still be affected in the future.
- *sane*: after the spread of the virus if over and no more nodes can be infected, the remaining susceptible nodes are said to be sane.

Initially ($t = 1$), all nodes are susceptible, except for a single node which is initially infected. Then, at each timeslot, the virus can propagate via communication links, from every infected nodes to its susceptible neighbors.

Let $p$ be a parameter of the system denoting the *infection probability*. At each round, an infected node attacks and attempts to infect each of its susceptible neighbors. For each attack, the susceptible node targeted is infected with probability $p$. Conversely, with probability $1 - p$, the node detects the attack, in which case it starts a containment strategy.

**Killing Strategies.** When a susceptible node is victim of a failed attack, it detects the attempt and can initiate a containment strategy. This paper considers a family of killing strategies, in which a certain number of nodes are killed to act as an obstacle to the spread.

We consider killing strategies that differ in the extent to which neighboring nodes are deactivated/killed. Strategy $Kh$-Hop is defined for any non-negative value of $h$, such that the detector (i.e., the node detecting the infection attempt) send messages to kill all nodes in its $h$-hops neighborhood, and then kills itself.

The behavior of a node receiving such a message depends on its current state:

- A susceptible node sacrifices itself as requested by the message, after having potentially relayed the message as requested by the strategy.
- An infected nodes ignores the message; it neither forward the message nor sacrifice itself.

– A killed node neither receives nor forwards the message, and cannot be infected.

When a node receives both a killing message and an infection message, there is a probability $\alpha$ that the strategy messages take precedence over infection messages. When $\alpha = 1$ we say that killing messages have priority.

An example of virus propagation and its containment by $K1$-Hop strategy is given in Appendix A.1.

**Infinite Grid.** The infinite grid corresponds to the graph $G = (V, E)$ where nodes are located on the square lattice. Each node is connected to the nodes at distance one in each of the four cardinal directions. $V = \mathbb{Z} \times \mathbb{Z}$ and $E = \{e_{i,j} | i \in V \wedge j \in V \wedge dist(i, j) = 1\}$.

## 4 Equivalence of Site Percolation and $K0$-Hop Strategy

In this section we show that site percolation and strategy $K0$-Hop have the same threshold, below which propagation is contained with probability one and above which it is not contained with non zero probability.

**Site Percolation.** Given an infinite graph and a probability $p$ such that any site (*i.e.*, node) is *occupied* with probability $p$ (resp., *empty* with probability $1 - p$), let us consider the random variable $X_p$ that equals 1 Equivalence of Site Percolation and $K0$-Hop Strategycomponent of occupied nodes and 0 otherwise.

There exists a threshold $\tau$ (called the percolation threshold) such that [2,14]:

$$\begin{cases} \mathbb{P}[X_p = 1] = 0 & \text{if } 0 \leq p < \tau \\ \mathbb{P}[X_p = 1] = 1 & \text{if } \tau < p \leq 1 \end{cases}$$

In other words, with probability 1, there is an infinite connected component of occupied nodes when the probability $p$ is above $\tau$. Respectively, there is no such component when $p$ is below the threshold. When $p$ exactly equals the threshold, the situation is unclear.

**Strategy $K0$-Hop.** Given a probability $p$ of infection and an infinite graph, consider a spread against the $K0$-Hop containment strategy. Let $Y_p$ be a random variable that equals 1 if the spread never stops and 0 otherwise.

For the two extreme values of $p$, the distribution of $Y_p$ is trivial: when $p = 0$, the spread is immediately contained and $\mathbb{P}[Y_0 = 1] = 0$; conversely, when $p = 1$, all nodes are infected and $\mathbb{P}[Y_1 = 1] = 1$. Moreover, by a simple coupling argument, it is also clear that the function $p \mapsto P[Y_p = 1]$ is non-decreasing. Therefore, there exists a unique threshold $\tau_0$ such that:

$$\begin{cases} \mathbb{P}[Y_p = 1] = 0 \text{ if } 0 \leq p < \tau_0 \\ \mathbb{P}[Y_p = 1] > 0 \text{ if } \tau_0 < p \leq 1 \end{cases}$$

Of course, it is also possible that either $\mathbb{P}[Y_p = 1] > 0$ for all $p > 0$ ($\tau_0 = 0$) or $\mathbb{P}[Y_p = 1] = 0$ for all $p < 1$ ($\tau_0 = 1$). It is important to note the inequality. Contrary to the random variable $X_p$ associated to site percolation, $Y_p$ does not "jump directly from 0 to 1". In fact, $Y_p$ equals 1 if and only if $p = 1$. For any probability of infection $p$ strictly lower than one, there is a non-zero probability that strategy $K0$-Hop contains the spread. Indeed, if all neighbors of the initially infected node detect the infection, the propagation stops immediately. This case happens with probability $(1-p)^\delta$, where $\delta > 0$ is the degree of the initial node, and thus $\mathbb{P}[Y_p = 1] \leq 1 - (1-p)^\delta$.

**Theorem 1.** *Site percolation and strategy $K0$ have the same thresholds: $\tau = \tau_0$.*

We can observe that, if the probability of infection of a node depends on the state of its neighbors (if the events are not independent), then the result may be different, but, with some bound on the probability of infection, we still obtain the following corollary.

**Corollary 1.** *Given a graph $G = (V, E)$, and let $\tau$ be the site percolation threshold on $G$. If, for all $u \in V$ and $t \in \mathbb{N}$, the probability $p_{u,t}$ that a susceptible node $u$ is infected at time $t$ is bounded by $\tau$, then the $K0$-Hop strategy contains the propagation of the virus with probability one.*

## 5 Virus Containment with Priority Killing Messages

In this section, we prove that it is always possible to contain the propagation in an infinite grid, provided that we use a killing strategy that sacrifices enough nodes (*i.e.*, a strategy $Kh$-Hop with $h$ large enough). The proof is done by reduction to strategy $K0$-Hop so that we can apply Theorem 1 and use the existence of a percolation threshold for a specific topology.

**Theorem 2.** *Given a virus spread with infection probability $p$, where $0 \leq p < 1$, there exists a killing strategy that contains the virus spread.*

### 5.1 Definitions and Explanations

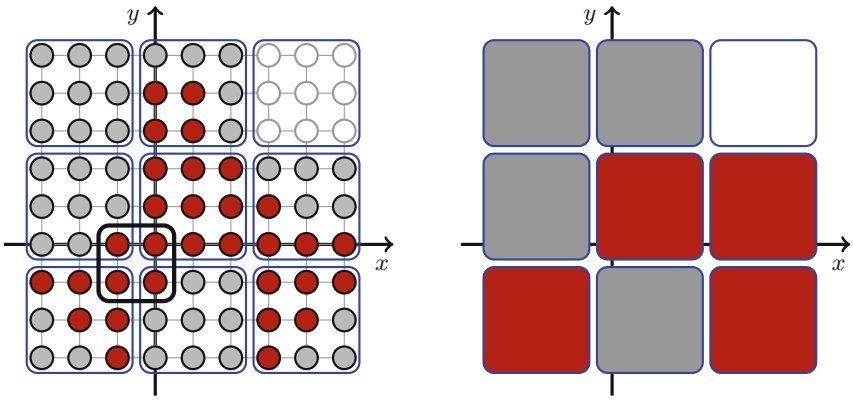
**General Idea of the Proof.** Based on the infinite grid, we define the notion of *super nodes* to encompass a squared subset of nodes. Super nodes are considered as infected, killed, or sane depending on the states of their internal nodes. By choosing an appropriate size for the super nodes and an adequate containment strategy, we can show that, at the super node level, the propagation of the virus behaves similarly as when strategy $K0$-Hop is executed. From there, we can deduce that the infection is contained at the super node level, which then implies containment at the lower level.

**Super Node.** Given a strictly positive integer $h$ and the infinite grid $G(V, E)$ with $V = \mathbb{Z} \times \mathbb{Z}$, we partition the nodes into *super nodes of size* $h$. Each super node is a subset of $h^2$ nodes organized in a square of side $h$. The super node at column $x$ and row $y$, denoted $g(x, y)$, corresponds to the following subset of $V$:

$$g(x, y) = \{(hx + i, hy + j) \in V \mid 0 \leq i, j < h\}$$

Similarly to nodes, super nodes may be infected, killed, or susceptible. We say a super node is infected if all nodes in at least one of its sides are infected. If a super node is not infected, we say it is killed if it contains at least a killed node, otherwise, we say it is susceptible (or sane if the spread is over). For simplicity, we assume that the super node containing the initial infected node (at $(0, 0)$ without loss of generality) is also infected.

**Containment Strategy.** Given a super nodes of size $h$, we study the strategy $K2h$-Hop. For the analysis we consider a slightly modified version of the $K2h$-Hop strategy: nodes do not send killing messages to neighbors belonging to different super nodes. Thus, the killing strategy affects only nodes contained in the same super node as the detector. This modification favors the propagation; it weakens the containment strategy and helps the virus propagate and therefore has no impact on our result. Indeed, if the spread is contained with the modified version, it is also contained with the standard $K2h$-Hop strategy.




(a) at node level (b) at super node level

**Fig. 1.** Example of infection

Figure 1 provides an example of infection at the nodes level and at the super nodes level. At the super node level, it is worth noting that, unlike what happens at node level, an infection may be transmitted in "diagonal": in the figure, the central super node has infected the super node located at the lower left corner.

Since an infection can be propagated via the diagonals, the infection in super nodes can be seen as an infection in the lattice where every node has 8 neighbors. This lattice is called Moore's neighborhood [16]. In this section, the neighbors of a super node are the neighbors according to Moore's neighborhood and the neighbors of normal node are the four neighbors in the default model (Von Neumann's neighborhood).

Also, one can observe that when a node in a super node $U$ detects an attack, then it broadcast killing messages in the whole super node (because all the nodes in the super node are at distance at most 2 h from one another). So that if one node is attacked, then all the other nodes in its super node are either killed or infected. Therefore, a super node is either sane, or contains no sane node.

**Small-Four.** Based on the previous observation, we introduce the notion of "Small Four". For any group of four super nodes arranged in a $2 \times 2$ grid, let us call "Small-Four" the group of four normal nodes that connect these four super nodes by their corners. In Fig. 1a, the group of four nodes $(0,0), (0,-1), (-1,0), (-1,-1)$ located in the small black square is an example of the "Small-Four" that connects the four super nodes $g(0,0), g(0,-1), g(-1,0), g(-1,-1)$.

**Rectangle.** In the remaining of the section, the distance $d(u,v)$ between two nodes $u$ and $v$, denotes the length of a shortest path between $u$ and $v$. For two nodes $u$ and $v$, $Rect(u,v)$ denotes the set of nodes in the rectangle delimited by $u$ and $v$: $Rect(u,v) = \{w \in V \mid d(u,w) + d(w,v) = d(u,v)\}$.

### 5.2 Proof

Now, we prove several Lemmas to analyze the propagation of the infection within a super node (and its origin), then we prove that the propagation of the virus at the super node level is a simple virus propagation in the square lattice with Moore's neighborhood against the $K0$-Hop strategy. Finally, we conclude by a proof of the theorem using the equivalence proved in the previous section.

**Lemma 1.** *Let $u$ be a node, in a super node $U$, infected for the first time at time $t$. Let $v$ be a node in $U$, at distance $d_v \geq 0$ from $u$. Then, $v$ is susceptible at time $t' < t - d_v$, and cannot be killed at time $t - d_v$.*

*Proof.* First, suppose by contradiction that there exists a node $v$ in $U$ and at distance $d_v$ from $u$ that detects a failed attack at time $t - d_v$. Then, the killing messages sent by $v$ reach $u$ at time $t$, killing node $u$ before it gets infected. So, each node in $U$ at distance $d$ from $u$ cannot detect an attack at time $t - d$ or before. Moreover, if a $v$ is infected at time $t - d_v - 1$, then the infection reaches $u$ before time $t$, a contradiction, so $v$ is susceptible at time $t' < t - d_v$.

**Lemma 2.** *Let $u$ be a node, in a super node $U$, infected for the first time at time $t$. Let $v$ be a node in $U$, at distance $d_v \geq 0$ from $u$, that is infected at time $t - d_v$. Then, each node in $Rect(u,v)$ and at distance $d$ from $u$ is infected at time $t - d$.*

*Proof.* Let $v$ be a node in $U$ at distance $d_v$ from $u$ that is at infected at time $t - d_v$. Let $w_1$ be a node in the rectangle $Rect(u, v)$ that is at distance 1 from $v$ (thus at distance $d_v - 1$ from $u$). By Lemma 1, we know that $w$ does not detect an infection at time $t - (d_v - 1)$, then the infection from $v$ is successful and $w_1$ is infected at time $t - (d_v - 1)$. Again, if $w_2$ is a node in $Rect(u, v)$ at distance 2 from $v$ then $w_2$ has a neighbor that is infected at time $t - (d_v - 1)$ (a node that is in $Rect(u, v)$ and at distance 1 from $v$). Therefore, $w_2$ is infected at time $t - (d_v - 2)$. Recursively, each node in $Rect(u, v)$ at distance $d$ from $u$ (and at distance $d_v - d$ from $v$) is infected at time $t - d$.

According to the previous Lemma, we can define formally what we mean by the origin of the infection of a node: the infection of a node $u$ *originates from a node* $v$ (in the same super as $u$) if $u$ is infected for the first time at time $t$ and each node in $Rect(u, v)$ and at distance $d$ from $u$ is infected at time $t - d$.

One can observe that the origin of the infection may not be unique as we can say the infection of $u$ may originates from any node in the rectangle $Rect(u, v)$. The interesting origin of the infection (in the super node) may be the first infected node with such property. Its clear that the first infected origin of the infection is on one side of the super node, and the next lemma proves that it is actually always a corner.

**Lemma 3.** *The infection of any node $u$, in a super node $U$, originates from a corner of $U$.*

*Proof.* The Lemma is proved by induction on the time $t$ of the first infection of $u$. At time 0, the lemma is proved for node $(0, 0)$ as it is in the corner itself.

Now let $u$ be a node infected for the first time at time $t$ and suppose the Lemma proved for every node that is infected before time $t$. Let $U$ be the super node of $u$. Let $v$ be one of the first infected node on the side of $U$ such that the infection of $u$ originates from $v$. If $v$ is first infected at time $t_v$, then it is a distance $t - t_v$ from $u$. Thus, all the neighbors of $v$ in $U$ are not infected at time $t_v - 1$ (otherwise, such neighbor would be an origin of the infection of $u$ in $U$, which contradicts the fact that $v$ is the first one). So that $v$ has a neighbor outside $U$, call it $w$, that is infected at time $t_w = t_v - 1$. By hypothesis, the infection of $w$ in its super node $W$ originates from a corner of $W$. For all possible corner $c$ of $W$, the rectangle $Rect(w, c)$ contains a corner $c_0$ of $W$ that is also a neighbor of a corner $b_0$ of $U$ (see Fig. 2). Let $d_b$ be the distance from $u$ to $b_0$ and $d_c$ be the distance from $c_0$ to $w$ (which is also the distance from $v$ to $b_0$). See Fig. 2 for an illustration of the configuration.

By Lemma 2, $c_0$ is infected at time $t_w - d_c$. So that at time $t_w - d_c + 1 = t_v - d_c$, node $b_0$ is either killed (it has detected the attack from $c_0$) or infected.

By the triangular inequality, we have $d_b \leq d_v + d_c$ so that $t_v - d_c = t - d_v - d_c \leq t - d_b$ and by Lemma 1, $b_0$ cannot be killed at time $t_v - d_c$. Therefore, the corner $b_0$ of $U$ is infected at time $t_v - d_c$. Since $b_0$ is at distance at most $d_v + d_c$ from $u$ then, by Lemma 2, the infection of $u$ originates from $b_0$. By hypothesis, $b_0$ cannot be infected before $v$ so $t_v - d_c = t_v$ i.e., $d_c = 0$, which means $v = b_0$ and $v$ is in a corner of $U$.

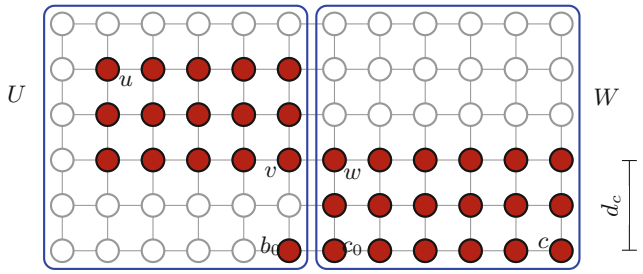



**Fig. 2.** The infection of a node $u$ originates from a corner of its super node.

**Lemma 4.** *The first infected node in a "Small-Four" belongs to an infected super node, or is the node* $(0,0)$ *and the source of the infection.*

*Proof.* Consider a "Small-Four" composed of the following set of nodes $S = \{(hx, hy), (hx-1, hy), (hx-1, hy-1), (hx, hy-1)\}$. Assume that $u \in S$ is the first infected node among the four nodes and belongs to the super node $U$. Because the three other nodes of $S$ are infected after node $u$, either $u = (0,0)$ is the source of the infection or $u$ is infected by another node inside $U$. In the latter case, according to Lemma 3, the infection of node $u$ originates from a corner of $U$ distinct from $u$ (because $u$ is infected by a node inside $U$). One can observe that the infection originates from a corner that is adjacent (i.e., on the same side of the super node) to the corner $u$. Indeed, if it originates from the opposite corner, it also originates by definition from any node in the super node, including the adjacent corners. This means that, at least one side of super node $U$ is entirely infected i.e., $U$ is infected.

**Lemma 5.** *A super node (distinct from* $g(0,0)$*) may be infected only if it is a direct or diagonal neighbor of a previously infected super node. Moreover, in this situation, the probability of being infected is bounded by* $4p^h$*.*

*Proof.* According to Lemma 3, if a super node $U$ is infected, the attack originates from a corner. The node $u$ in this corner was infected by a neighbor outside $U$, thus by a node that belongs to the same "Small-Four". By Lemma 4, the first infected node of this "Small-Four" belongs to an infected super node, or to $g(0,0)$ (which is considered infected). Hence, one neighbor (direct or diagonal) of super node $U$ is infected.

Let us recall that $p$ is the probability that a normal node in the grid is infected. To become infected, a super node must have one of its sides entirely infected. For a given side, the probability that it becomes infected is exactly by $p^h$ since any of the $h$ nodes has a probability $p$ of being infected. The events corresponding to the entire infection of each side are not independent (two adjacent sides have a node in common), but the probability that a super node has at least one side entirely infected is bounded by $4p^h$.

In more details the probability that a super node is infected depends on the way its neighbors are infected. Indeed, when considering a super node $U$, if only

one side of a neighbor is infected, and if this side is not connected to $U$ then $U$ is not attacked by this neighbor and the probability that $U$ is infected is 0. But, if a unique node effectively attacks $U$, then the probability that $U$ is infected is exactly $2p^h - p^{2h-1}$, which is the probability that at least one of the two sides of $U$ adjacent to the attacked corner gets entirely infected. Since $U$ can be attacked from different corner at different time, we simply bound the probability that $U$ is infected by $4p^h$.

*Proof (Proof of Theorem 2).* According to Lemma 5, the set of infected super nodes forms a single connected component, with the Moore's neighborhood (two diagonal super nodes are considered connected).

Since every neighbor of an infected super node has a bounded probability to be infected, the infection spreading through super nodes can be seen as strategy $K0$ on an infinite lattice with Moore's neighborhood.

According to Corollary 1, if we choose $p$ such that the probability that a node gets infected is smaller than the percolation threshold, then the propagation of the virus is contained by the $K0$-Hop with probability one. According to Malarz and Galam [16], there exists a percolation threshold in an infinite lattice with Moore's neighborhood. Let $\rho_c^{\text{Moore}}$ be the percolation threshold in an infinite lattice with Moore's neighborhood.

Let $h \in \mathbb{N}$ be such that $h > \frac{\log(\rho_c^{\text{Moore}}/4)}{\log p}$. The virus spreading with infection probability $p$ against the $K2h$-Hop strategy can be seen as the propagation of virus via super nodes of size $h$ with probability bounded by $4p^h < \rho_c^{\text{Moore}}$ against strategy $K0$-Hop. By Corollary 1 such propagation is contained, so that the virus propagation with infection probability $p$ is contained by strategy $K2h$-Hop in an infinite grid.

## 6 The Case of Non-Priority Killing Message

When the killing messages do not have priority ($\alpha < 1$) then the analysis of the previous section does not hold. In this Section we study the behavior of the best possible strategy, which is $K\infty$-Hop. Even though this strategy may not be applied in practice, as all nodes in the network are killed if a virus is detected, it has theoretical interest. Indeed, if the virus is not contained when using the $K\infty$-Hop strategy, then, no strategy contains the virus.

First, one can observe that, since we consider the $K\infty$-Hop strategy, at time-slot $t \in \mathbb{N}$ all the nodes at distance $t$ from the origin are either infected or killed. Those nodes form a square whose vertices are nodes $(t, 0)$, $(0, t)$, $(-t, 0)$ and $(0, -t)$. Therefore, we can restrict our analysis on a quarter of the grid: if the virus is contained with probability one (resp. smaller than one) on a quarter of the grid, it is contained with probability one (resp. smaller than one) on the whole grid. Also, we can order the nodes and say that the two predecessors of a node are its neighbors that are closer to the origin. Figure 3 represents an execution of $K\infty$-Hop starting from an initial infected node at $(0, 0)$.

We know from the previous Section that, when $\alpha = 1$, $K\infty$-Hop strategy successfully contains the virus. However, when $\alpha = 0$, it depends on the probability $p$. Indeed, when $\alpha = 0$, a node becomes infected if at least one of its

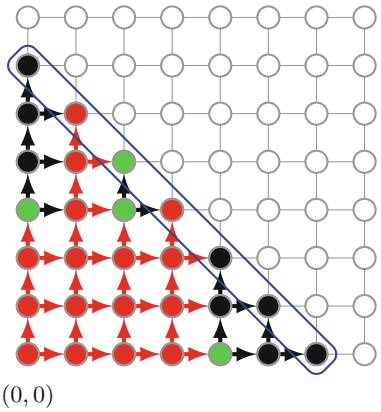



**Fig. 3.** An execution of $K\infty$-Hop on the upper right quarter of an infinite grid. Red, green, and black nodes are resp. infected, detector, and killed. (Color figure online)

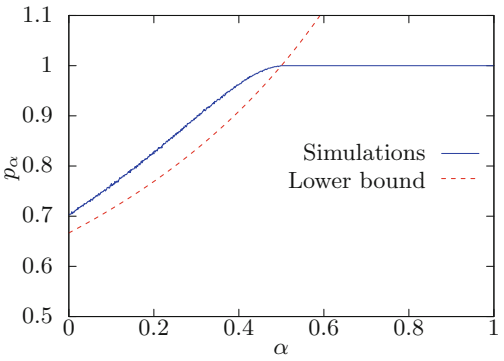



**Fig. 4.** Threshold of the probability $p_\alpha$ to contain the virus, depending on the value of $\alpha$.

two predecessors is infected. This model is exactly a directed site percolation on the square lattice. The percolation threshold for this model is around 0.7054852 [8,9,26] (computed by simulations). Thus $K\infty$-Hop strategy contains the virus if $p$ is above this threshold. Now, the main result of this Section, is to give a lower bound on the the value of the percolation threshold $p_\alpha$, depending on the value of $\alpha$. Figure 4 depicts the lower bound and an approximation of the threshold that we computed by simulations.

**Theorem 3.** *If* $p < \min\left(1, \frac{2}{3-2\alpha}\right)$, *the virus is contained. In particular, the virus is always contained if* $\alpha \geq 0.5$ *and* $p \neq 1$.

*Proof.* On Fig. 3, the blue box encompasses the nodes that were infected/killed at round 6. We are interested in the evolution of the nodes in the diagonal over the time, see Fig. 5. One can observe that if there are $k$ killed nodes at one extremity of the diagonal, then there will be also at least $k$ killed nodes on the same extremity of the next diagonal.

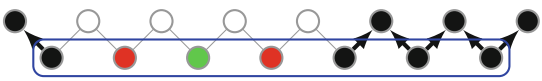


**Fig. 5.** The spread of the virus on the diagonal ($k_1 = 1$ and $k_2 = 3$)

Consider a diagonal of size $d$, which contains $k_1$ killed nodes at one extremity and $k_2$ at the other. Let us define the central part of the diagonal as the $n =$

$d - k_1 - k_2$ nodes between the two extremal blocks of killed nodes. At the next round of the execution, the new diagonal contains $d + 1$ nodes and its central part contains $n - 1$, $n$, or $n + 1$ nodes.

The proof of the theorem is the result of the two following Claims, that are true for an arbitrary constant $X \in \mathbb{N}$:

- Claim 1: if the central part has at most $2X$ nodes, the probability the virus is contained in the next round is greater than a fixed constant $C_X > 0$.
- Claim 2: if the central part has more than $2X$ nodes and if $p < \frac{2}{3-2\alpha+f(X)}$ (with $\lim_{X\to\infty} f(X) = 0$), then eventually the central part has size at most $2X$.

Indeed, if $p < \frac{2}{3-2\alpha}$ we can choose $X$ large enough so that $p < \frac{2}{3-2\alpha+f(X)}$. Then, by the second Claim, there cannot be an infinite execution without the condition of the first Claim being verified, so that we always have a strictly positive probability that the virus is contained i.e., the virus is always contained.

The proof of the first Claim is straightforward as there is a probability at least $C_X = (1-p)^{2X+1} > 0$ that all nodes on the next diagonal detect the virus.

The proof of the second Claim is a by studying the expectation $E$ of the reduction of the size of the central part after one round, on one side of the diagonal (since the other side has the same behavior).

The central part decreases by $k$ nodes when the node in the extremity is killed (not infected or receives the killing message from one of its neighbors), $k - 1$ other nodes detect the attack and the next node is infected. This event occurs with probability $(1 - (1 - \alpha)p)(1 - p)^{k-1}p$. We consider only the case $k \leq X$ because we are on one side of a diagonal of length at least $2X$. Then

$$E = (1 - (1 - \alpha)p)\ p \sum_{k=0}^{X} k(1-p)^{k-1}.$$

For $p < 1$, $\sum_{k=0}^{\infty} k(1-p)^{k-1} = \frac{1}{p^2}$, therefore $E = (1 - (1 - \alpha)p)\ p \left(\frac{1}{p^2} - g(X)\right)$

with $\lim_{X\to\infty} g(X) = 0$. The size $d$ of the diagonal increases by 1 every round. Since we considered only one side of the diagonal, we need to solve $E > 1/2$ to determine when containment is possible, which gives the following bound:

$$p < \frac{2}{3 - 2\alpha + f(X)} \qquad \text{with } \lim_{X\to\infty} f(X) = 0$$

## 7 Conclusion

We considered the virus propagation problem where nodes that detect a failed infection attempt can broadcast a deactivation message to their neighbors to contain the propagation. We first show that the strategy consisting in deactivating itself when an attack is detected (without alerting its neighbors) has the

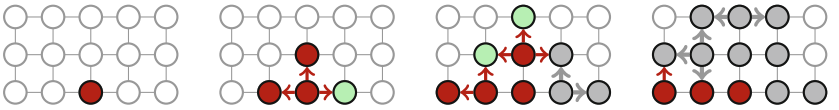
same behavior as the site percolation, in the sense that both problems have the same critical probability threshold. Then, we prove that in an infinite grid graph the propagation of a virus can always be contained when the killing message have priority, for all value of $0 \leq p < 1$, by choosing an adequate strategy. Finally, when killing messages do not have priority, we showed a lower bound on the threshold of the infection probability, above which the virus spreads infinitely with a positive probability.

**Acknowledgments.** This work is partially supported by JSPS KAKENHI Grant (C)(JP15K00183) and (JP15K00189) and Japan Science and Technology Agency, CREST (JPMJCR1404) and Infrastructure Development for Promoting International S&T Cooperation and Project for Establishing a Nationwide Practical Education Network for IT Human Resources Development, Education Network for Practical Information Technologies.

# A Appendix

## A.1 *K*1-Hop Strategy Example

Figure 6 depicts a step-by-step execution of strategy $K1$-Hop in a $3 \times 5$ grid. An infected node attacks its neighbors at timeslot $t$; the right neighbor detects the attack, the left and the top ones become infected. Following strategy $K1$-Hop, the right neighbor (detector) sends kill messages to its own neighbors and kills itself; its two neighbors (top and right) react accordingly, even though the top one is attacked at the same time. Two other nodes detect the attack at time $t+1$. At time $t+2$, one of the kill message reaches an infected node and is ignored. The state of the nodes at time $t+2$ is final, and the network is partitioned i.e., the spread is contained.



(a) Timeslot $t-1$ (b) Timeslot $t$ (c) Timeslot $t+1$ (d) Timeslot $t+2$

**Fig. 6.** Example of the spread of a virus against the strategy $K1$-Hop

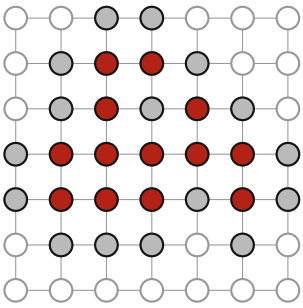
## A.2 Omitted Proof

**Theorem 1 (restated).** *Site percolation and strategy $K0$ have the same thresholds: $\tau = \tau_0$.*

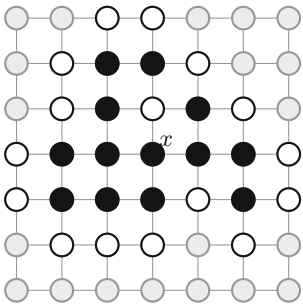
*Proof.* The proof consists in showing that $Y_p$ and $Z_p^x$ follow the same distribution for any $p$ and any occupied node $x$. To prove the claim, consider the following (intuitive) link between both models in Fig. 7a.

| strategy $K0$ | site-percolation |
|---|---|
| infected node | occupied node |
| killed node | empty node |
| sane node | — *(no equivalent)* |
| initial infected node | any occupied node $x$ |

(a) Intuitive link between $K0$ and site-percolation



(b) Strategy $K0$-Hop




(c) Site-percolation

**Fig. 7.** "Equivalence" between strategy $K0$-Hop and site-percolation model (Color figure online)

With site percolation, *all* nodes are randomly set to the occupied state (with probability $p$) or to the empty state (with probability $1-p$). With strategy $K0$-Hop, *some* nodes are randomly set to the infected state (with probability $p$) or to the killed state (with probability $1-p$) , but *some other* nodes are not set in one of these states. Indeed, any node that is neither an infected node, nor a neighbor of an infected node, remains in a sane state (see Sect. 3).

However, sane nodes do not have any effect on the random variable $Y_p$ since they are disconnected from the initially infected node. The value of $Y_p$ depends only on the infected (red nodes on Fig. 7b) and killed nodes (gray nodes): all infected nodes are connected and therefore $Y_p$ reflects the infiniteness of this infected component.

With site percolation, for a given occupied vertex $x$, the same observation applies to the random variable $Z_p^x$. It depends only on the occupied nodes (black nodes on Fig. 7c) that are connected to $x$ or the empty nodes (white nodes) that are around these occupied nodes. Other nodes (light gray nodes) do not have any effect on $Z_p^x$; they could be occupied or empty; it will not change the value of $Z_p^x$.

Since (1) nodes are infected or occupied with the same probability $p$ starting from an initial infected node or an arbitrary occupied node $x$, and (2) $Y_p = Z_p^x$, it follows that $\tau = \tau_0$.

## References


1. Anderson, R.M., May, R.M.: Population biology of infectious diseases: part I. Nature **280**(5721), 361–367 (1979)
2. Bollobás, B., Riordan, O.: Percolation. Cambridge University Press, Cambridge (2006)
3. Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards automatic generation of vulnerability-based signatures. In: Proceedings of the 27th IEEE Symposium on Security and Privacy, S&P 2006, pp. 2–16, May 2006
4. Chakrabarti, D., Wang, Y., Wang, C., Leskovec, J., Faloutsos, C.: Epidemic thresholds in real networks. ACM Trans. Inf. Syst. Secur. **10**(4), 1 (2008)
5. Falliere, N.: Sality: story of a peer-to-peer viral network, July 2011. http://www.symantec.com/connect/sites/default/files/sality_peer_to_peer_viral_network.pdf
6. Grassberger, P.: On the critical behavior of the general epidemic process and dynamical percolation. Math. Biosci. **63**(2), 157–172 (1983)
7. Hu, R., Sopena, J., Arantes, L., Sens, P., Demeure, I.: Fair comparison of gossip algorithms over large-scale random topologies. In: Proceedings of the 31st IEEE International Symposium on Reliable Distributed Systems (SRDS 2012), pp. 331–340, October 2012
8. Jensen, I.: Low-density series expansions for directed percolation: II. The square lattice with a wall. J. Phy. A: Math. Gen. **32**(33), 6055 (1999). https://stacks.iop.org/0305-4470/32/i=33/a=304
9. Jensen, I.: Low-density series expansions for directed percolation: III. Some two-dimensional lattices. J. Phy. A: Math. Gen. **37**(27), 6899 (2004). http://stacks.iop.org/0305-4470/37/i=27/a=003
10. Kephart, J.O., White, S.R.: Directed-graph epidemiological models of computer viruses. In: Proceedings of the 12th IEEE Symposium on Security and Privacy, S&P 1991, pp. 343–361 (1991)
11. Kephart, J.O., White, S.R.: Measuring and modeling computer virus prevalence. In: Proceedings of the 14th IEEE Symposium on Security and Privacy, S&P 1993, pp. 2–15 (1993)
12. Kermack, W.O., McKendrick, A.G.: A contribution to the mathematical theory of epidemics. In: Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character, vol. 115, no. 772, pp. 700–721 (1927)
13. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 207–226. Springer, Heidelberg (2006). https://doi.org/10.1007/11663812_11
14. Lee, M.J.: Pseudo-random-number generators and the square site percolation threshold. Phy. Rev. E **78**(3), 031131 (2008)
15. Li, Z., Sanghi, M., Chen, Y., Kao, M.Y., Chavez, B.: Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In: Proceedings of the 27th IEEE Symposium on Security and Privacy, S&P 2006, pp. 32–47, May 2006
16. Malarz, K., Galam, S.: Square-lattice site percolation at increasing ranges of neighbor bonds. Phys. Rev. E **71**, 016125 (2005). https://doi.org/10.1103/PhysRevE.71.016125
17. May, R.M., Anderson, R.M.: Population biology of infectious diseases: part II. Nature **280**(5722), 455–461 (1979)

18. Moore, D., Shannon, C., Voelker, G., Savage, S.: Internet quarantine: requirements for containing self-propagating code. In: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2003, vol. 3, pp. 1901–1910, Mar 2003
19. Nguyen, T.D., Bonnet, F., Défago, X.: Analyzing the impact of mitigation strategies on the spread of a virus. Research Report IS-RR-2014-002, Japan Advanced Institute of Science and Technology (JAIST), May 2014
20. Pastor-Satorras, R., Vespignani, A.: Epidemic dynamics and endemic states in complex networks. Phys. Rev. E **63**(6), 066117 (2001)
21. Pastor-Satorras, R., Vespignani, A.: Epidemic spreading in scale-free networks. Phys. Rev. Lett. **86**, 3200–3203 (2001)
22. Sasson, Y., Cavin, D., Schiper, A.: Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In: Proceedings of the IEEE Wireless Communications and Networking, WCNC 2003, pp. 1124–1130 (2003)
23. Staniford, S., Paxson, V., Weaver, N.: How to own the internet in your spare time. In: Proceedings of the 11th USENIX Security Symposium, USENIX-Security 2002, pp. 149–167. USENIX Association, Berkeley, CA, USA (Aug 2002)
24. Van Mieghem, P.: The $N$-intertwined SIS epidemic network model. Computing **93**(2–4), 147–169 (2011)
25. Van Mieghem, P., Omic, J., Kooij, R.E.: Virus spread in networks. IEEE/ACM Trans. Networking **17**(1), 1–14 (2009)
26. Wang, J., Zhou, Z., Liu, Q., Garoni, T.M., Deng, Y.: High-precision Monte Carlo study of directed percolation in $(d+1)$ dimensions. Phys. Rev. E **88**, 042102 (2013). https://doi.org/10.1103/PhysRevE.88.042102
27. Xia, J., Vangala, S., Wu, J., Gao, L., Kwiat, K.: Effective worm detection for various scan techniques. J. Comput. Secur. **14**(4), 359–387 (2006)
28. Xu, W., Zhang, F., Zhu, S.: Toward worm detection in online social networks. In: Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC 2010, pp. 11–20. ACM, New York, December 2010
29. Zhou, L., Zhang, L., McSherry, F., Immorlica, N., Costa, M., Chien, S.: A first look at peer-to-peer worms: threats and defenses. In: Castro, M., van Renesse, R. (eds.) IPTPS 2005. LNCS, vol. 3640, pp. 24–35. Springer, Heidelberg (2005). https://doi.org/10.1007/11558989_3
30. Zou, C.C., Gong, W., Towsley, D.: Code red worm propagation modeling and analysis. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, pp. 138–147. ACM, New York (2002)
31. Zou, C.C., Towsley, D., Gong, W.: Modeling and simulation study of the propagation and defense of internet e-mail worms. IEEE Trans. Dependable Secure Comput. **4**(2), 105–118 (2007)

# Computational Complexity

# Improved Distributed Algorithms for Coloring Interval Graphs with Application to Multicoloring Trees

Magnús M. Halldórsson[1] and Christian Konrad[2(✉)]

[1] ICE-TCS, School of Computer Science, Reykjavik University, Reykjavik, Iceland
mmh@ru.is
[2] Department of Computer Science,
Centre for Discrete Mathematics and its Applications (DIMAP),
University of Warwick, Coventry, UK
c.konrad@warwick.ac.uk

**Abstract.** We give a distributed $(1+\epsilon)$-approximation algorithm for the minimum vertex coloring problem on interval graphs, which runs in the $\mathcal{LOCAL}$ model and operates in $O(\frac{1}{\epsilon}\log^* n)$ rounds. If nodes are aware of their interval representations, then the algorithm can be adapted to the $\mathcal{CONGEST}$ model using the same number of rounds. Prior to this work, only constant factor approximations using $O(\log^* n)$ rounds were known [12]. Linial's ring coloring lower bound implies that the dependency on $\log^* n$ cannot be improved. We further prove that the dependency on $\frac{1}{\epsilon}$ is also optimal.

To obtain our $\mathcal{CONGEST}$ model algorithm, we develop a color rotation technique that may be of independent interest. We demonstrate that color rotations can also be applied to obtain a $(1+\epsilon)$-approximate multicoloring of directed trees in $O(\frac{1}{\epsilon}\log^* n)$ rounds.

## 1 Introduction

Vertex coloring problems are central in distributed computing. Given a graph $G = (V, E)$, the objective is to compute an *s-coloring* $\gamma : V \to \{1, 2, \ldots, s\}$ in a distributed fashion, for an integer $s$, i.e., to assign each vertex one of $s$ colors so that adjacent nodes receive different colors. A substantial amount of research has been carried out on computing $(\Delta + 1)$-colorings, where $\Delta$ is the maximum degree of the input graph. This is an attractive bound, since it is easy to see that $\Delta + 1$ colors always suffice to color a graph. The quantity $\Delta + 1$ may however be an arbitrarily poor approximation of the *chromatic number* $\chi(G)$ of a graph $G$, which is the minimum number of colors needed in any coloring[1]. In

M. M. Halldórsson is supported by grants 152679-05 and 174484-05 from the Icelandic Research Fund. C. Konrad is supported by the Centre for Discrete Mathematics and its Applications (DIMAP) at Warwick University and by EPSRC award EP/N011163/1.

[1] For example a complete bipartite graph $G = (A, B, E)$ with $|A| = |B| = n$ can be colored with 2 colors while $\Delta = n$.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 247–262, 2017.
https://doi.org/10.1007/978-3-319-72050-0_15

this paper, we are therefore interested in distributed approximation algorithms for the *minimum vertex coloring problem*, which asks for a coloring with $\chi(G)$ colors.

**Computational Models.** We consider the $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ models of distributed computation. The input graph $G = (V, E)$ models a communication network, where computational units are located at every node $v \in V$. Graph $G$ also constitutes the problem input. The goal of a distributed coloring algorithm is to compute a (global) coloring, where every node reports its color upon termination of the algorithm. Every node $v \in V$ has a unique identifier $\mathrm{ID}(v)$ and, initially, besides their identifiers, nodes are aware of the identifiers of their neighbors (and hence also of its degree). Messages are exchanged in synchronous communication rounds, where a node may exchange individual messages with each of its neighbors. In the $\mathcal{LOCAL}$ model, messages of unbounded sizes may be exchanged, while in the $\mathcal{CONGEST}$ model, message sizes are limited to $\mathrm{O}(\log n)$, where $n$ is the number of nodes in the input graph. In both models the objective is to minimize the number of communication rounds required to complete the algorithm.

**Minimum Vertex Coloring Problem.** On general graphs, the minimum vertex coloring problem is NP-complete [14] and even hard to approximate within a factor of $n^{1-\epsilon}$ [23]. Nevertheless, since many distributed models focus on the number of communication rounds rather than the runtime of individual network nodes, it is possible to compute a $O(n^{\epsilon})$-approximation in $\exp\left(O(\frac{1}{\epsilon})\right)$ communication rounds on general graphs [5]. Linial presented a lower bound showing that coloring unoriented $d$-regular trees with $o(\sqrt{d})$ colors[2] requires $\Omega(\log_d n)$ rounds [16]. This result shows that for every graph class that contains trees, computing a $C$-approximation requires $\Omega(\log n)$ rounds, for every constant $C$.

**Multicoloring.** There are conceptual links between graph coloring and graph *multicoloring*:

**Definition 1.** *Let $G = (V, E, w)$ be a graph with vertex weights $w : V \to \mathbb{N}$. For an integer $k \geq 1$, a* $k$-multicoloring *of $G$ is an assignment $\phi : V \to 2^{[k]}$ such that:*

1. *For every $v \in V : |\phi(v)| = w(v)$, and*
2. *For every pair of adjacent vertices $u, v \in V : \phi(u) \cap \phi(v) = \varnothing$.*

The multichromatic number $\chi^m(G)$ is the largest number of colors needed in every multicoloring of $G$. In the *minimum vertex multicoloring problem*, the goal is to find a multicoloring that uses $\chi^m(G)$ colors. Distributed algorithms for graph multicoloring find applications in computing MAC schedules (see [15] and the references therein). Kuhn [15] studied a distributed multicoloring problem on general graphs, where a node $v$ of degree $\deg(v)$ receives a $(1-\epsilon)\frac{1}{\deg(v)+1}$ fraction of all colors. Similar to the notion of a $(\Delta+1)$-coloring, this quality

[2] Chang et al. argue in [8] that using a graph construction by Bollobás [7], the same lower bound holds even for colorings with $o(d/\log d)$ colors.

bound is given by the degrees of the vertices alone and may be arbitrarily far from an optimal multicoloring.

**Results.** In this paper, we primarily address the minimum vertex coloring problem in *interval graphs*, which are the intersection graphs of intervals on the line. Interval graphs do not contain trees and the previously mentioned lower bound thus does not apply. In a previous work, we gave a distributed 8-approximation algorithm that runs in $O(\log^* n)$ in the $\mathcal{LOCAL}$, and an extension of the algorithm to the $\mathcal{CONGEST}$ model if the representation of the intervals are known by the network nodes [12]. We improve on [12] and give distributed $(1+\epsilon)$-approximation algorithms for coloring interval graphs for both the $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ models, which run in $O(\frac{1}{\epsilon}\log^* n)$ rounds. Similar to [12], our $\mathcal{CONGEST}$ model algorithm requires that nodes are aware of their interval boundaries.

Our $\mathcal{CONGEST}$ model algorithm uses a color rotation technique that may be of independent interest. In the full version, we leverage this color rotation technique to give a $(1+\epsilon)$-approximation algorithm for multicoloring directed trees, running in in $O(\frac{1}{\epsilon}\log^* n)$ rounds in the $\mathcal{CONGEST}$ model. This is the first distributed algorithm for multicoloring problems with non-trivial approximation guarantee. Our lower bound construction also shows that obtaining a $(1+\epsilon)$-approximation of multicoloring paths requires $\Omega(1/\epsilon)$ rounds.

**Techniques.** The $\mathcal{LOCAL}$ model algorithm of [12] simulates the sequential Greedy coloring algorithm, which traverses the vertices in an arbitrary order and assigns the smallest color possible. The approximation guarantee of their algorithm follows from the fact that a Greedy coloring of interval graphs gives an 8-approximation [21]. They construct a dominating set, which can be colored using few colors, via a somewhat technical algorithm.

Our strategy is arguably simpler. We first compute a maximal distance $k$-independent set $I$ ($k = \Theta(\frac{1}{\epsilon})$) using an algorithm of Schneider and Wattenhofer [19]. Then, nodes of $I$ color their inclusive neighborhoods optimally. Notice that every inclusive neighborhood $\Gamma[v]$ of a vertex $v$ is a separator in interval graphs. The set of yet uncolored nodes thus form connected components of diameter $\Theta(k)$. Using a theorem about the chromatic number of circular arc graphs by Valencia-Pabon [22], we show that there exists a completion of the current partial coloring to a $(1+\epsilon)$-approximate coloring of the entire graph. Nodes of $I$ then color the connected components of uncolored nodes using the optimal color completion.

This approach relies heavily on the ability to send messages of unbounded sizes. When nodes of $I$ color connected components of uncolored nodes, they first need to collect the topology of entire components, which cannot be done in the $\mathcal{CONGEST}$ model. To overcome this difficulty, we develop a *color rotation* technique: Let $u, v \in I$ be nodes of the distance-$k$ maximal independent set such that $u$ is located left of $v$ and there is no other node of $I$ between them (the distance between $u$ and $v$ is thus at most $2k$). Suppose that their local neighborhoods have already been colored optimally. We show that a Greedy left-to-right coloring sweep can be initiated at $u$ that respects the colors of

$u$'s neighborhood and colors all nodes between $u$ and $v$. This coloring however does not necessarily respect the colors of $v$'s neighborhood. Similarly, a right-to-left coloring with similar properties is initiated by $v$. Since a Greedy coloring that processes the vertices with increasing left boundaries (or decreasing right boundaries) gives an optimal coloring in interval graphs, the two coloring sweeps produce optimal colorings. We then apply our color rotation technique: Guided by the right-to-left coloring, we perform color rotations using few additional colors that transform the left-to-right coloring into a coloring that respects the colors of the neighborhood of $v$, giving a $(1+\epsilon)$-approximation.

We demonstrate that the color rotation technique can be applied for multi-coloring directed trees as well. Our algorithm first computes a partitioning of the input tree into subtrees, which are then colored independently. The potential color conflicts between subtrees are then resolved via color rotations. In order to obtain the partitioning of the input tree into subtrees, we develop an algorithm for computing a particular *ruling set*, which may be of independent interest.

**Further Related Work.** To our best knowledge, the only works that explicitly address distributed algorithms for the minimum vertex coloring problem are the already mentioned algorithms by Barenboim et al. [5] for general graphs and our previous work on coloring interval graphs [12]. Goldberg et al. [11] gave a 7-coloring algorithm of planar graphs, which runs in $O(\log n)$ rounds, and a 5-coloring algorithm, which runs in $O(\log n \log\log n)$ rounds and requires the planar representation of the input graph. Barenboim and Elkin [3] gave a $(\lfloor(2+\epsilon)a\rfloor+1)$-coloring algorithm for graphs of arboricity $a$, which runs in $O(a \log n)$ rounds and thus subsumes the previously mentioned 7-coloring algorithm (planar graphs have arboricity at most 3). Schneider et al. gave a $((1-O(\chi(G)))\Delta)$-coloring algorithm whose runtime depends on the chromatic number $\chi(G)$ [18].

Due to the similarity in the problem statement, we also expand on distributed $(\Delta+1)$-coloring algorithms: The first randomized $(\Delta+1)$-coloring algorithm uses a reduction to the maximal independent set problem given by Luby [17]. Since the maximal independent set problem can be solved in $O(\log n)$ time via the algorithms of Luby [17] or Alon et al. [1], a $O(\log n)$ rounds algorithm is obtained. Improved randomized algorithms were given by Schneider and Wattenhofer [20], which runs in $O(\log\Delta + \sqrt{\log n})$ rounds, and later by Barenboim et al. [6], which runs in $O(\log\Delta)+2^{O(\sqrt{\log\log n})}$ rounds. Very recently, the first randomized algorithm, which runs in $o(\log n)$ rounds for any value of $\Delta$, was presented. Harris et al. showed that $O(\sqrt{\log\Delta})+2^{O(\sqrt{\log\log n})}$ suffice [13]. Deterministic $(\Delta+1)$-coloring algorithms have been extensively studied as well. The currently fastest algorithm is by Fraigniaud et al. [10] and uses $O(\sqrt{\Delta}\log^{2.5}\Delta+\log^* n)$. This result improved on Barenboim's algorithm [2], which uses $O(\Delta^{\frac{3}{4}}\log\Delta+\log^* n)$ rounds and was the first deterministic $(\Delta+1)$-coloring algorithm which achieved a sublinear in $\Delta$ number of rounds. Faster $(\Delta+1)$-colorings can be achieved on special graph classes. The well-known Cole-Vishkin algorithm [9] colors cycles (and directed trees) using 3 colors in $O(\log^* n)$ rounds, which is best possible due to a lower bound given by Linial [16]. This algorithm has been extended to bounded-independence graphs by Schneider and Wattenhofer [19]. For further

references, we refer the reader to the survey by Barenboim and Elkin [4] and the references therein.

**Outline.** We give notations and definitions in Sect. 2. Then we present our coloring algorithms for interval graphs in Sect. 3 and our lower bound in Sect. 4. Finally, we conclude in Sect. 5.

## 2 Preliminaries

**Definitions.** A *distance-k independent set* in a graph $G = (V, E)$ is a subset of vertices $I \subseteq V$ such that every pair of vertices $v_1, v_2 \in I$ is at distance at least $k$. A distance-$k$ independent set $I$ is *maximal* if $I \cup v$ is not a distance-$k$ independent set, for all $v \in V \setminus I$. We call a distance-2 independent set simply *independent set*. For an integer $k$, a *distance-k-coloring* of a graph $G = (V, E)$ is an assignment $\gamma : V \to \{1, \ldots, s\}$ of $s$ colors to the vertices such that every pair of vertices at distance at most $k$ receives different colors. A *partial coloring* of a graph $G = (V, E)$ is an assignment $\gamma : V \to \{1, \ldots, s\} \cup \{\perp\}$, where uncolored nodes are assigned the symbol $\perp$.

For simplicity, we assume that the input graphs are connected. The *neighborhood* of a vertex $v$ in graph $G$ is denoted by $\Gamma_G(v)$, and we define the inclusive neighborhood of $v$ by $\Gamma_G[v] = \Gamma_G(v) \cup \{v\}$. For a subset $V' \subseteq V$, we write $\Gamma_{V'}(v)$ to denote $\Gamma_G(v) \cap V'$. Furthermore, the *k-neighborhood* of a vertex $v$ is the set of nodes that are within distance at most $k$ from $v$ (excluding $v$), and we denote it by $\Gamma^k_G(v)$. Then $\Gamma^1_G(v) = \Gamma_G(v)$. For a vertex $v \in V$, we denote by $\deg_G(v)$ the degree of $v$ in $G$. For a subset $V' \subseteq V$, we may also write $\deg_{V'}(v)$ for the degree of $v$ in the subgraph of $G$ induced by the nodes $V'$, that is, $\deg_{V'}(v) := \deg_{G|_{V'}}(v)$.

**Interval Graphs.** Let $V = \{v_1, \ldots, v_n\}$ be a set of intervals with $v_j = (a_j, b_j)$ for all $1 \leq j \leq n$ and real numbers $a_j, b_j$ such that $a_j < b_j$. We assume that all $a_i, b_i$ are distinct. Let $G = (V, E)$ be the corresponding interval graph, i.e., there is an edge between vertices (intervals) $v_j, v_k$ if the two intervals intersect. We denote the number of edges by $m$.

We say that an interval $v$ is *proper* if no other interval $u$ satisfies $\Gamma_G[v] \subsetneq \Gamma_G[u]$. For an interval graph $G = (V, E)$, we denote by $G_P = (V_P, E|_{V_P})$ the subgraph of $G$ induced by the proper intervals of $G$. It is easy to see that $G_P$ is connected if $G$ is connected as well.

Graph $G_P$ is of bounded-independence, a property that restricts the sizes of maximum independent sets in local neighborhoods, formally defined as follows:

**Definition 2 (Bounded-independence Graphs).** *A graph $G = (V, E)$ is of* bounded-independence *if there is a bounding function $f(r)$ such that for each node $v \in V$, the size of a maximum independent set in the $r$-neighborhood of $v$ is at most $f(r), \forall r \geq 0$.*

Schneider and Wattenhofer [19] gave a distributed maximal independent set algorithm for graphs of bounded-independence that runs in time $O(\log^* n)$. We denote this algorithm by MisBI. It can be implemented in the $\mathcal{CONGEST}$ model.

## 3 Algorithms for Coloring Interval Graphs

We first give our algorithm for the $\mathcal{LOCAL}$ model in Sect. 3.1 and then show how to extend the algorithm to the $\mathcal{CONGEST}$ model in Sect. 3.2.

### 3.1 Algorithm in the $\mathcal{LOCAL}$ model

Our algorithm is depicted and explained in Algorithm 1. It is parametrized by an integer $k$, which determines the approximation guarantee and whose precise value is determined later.

**Algorithm 1.** Algorithm for Coloring Interval Graphs in the $\mathcal{LOCAL}$ Model

1. **Identify the subgraph $G_P$ of proper intervals:** Each node $v$ checks if it has a neighbor $u$ with $\Gamma_G[u] \supsetneq \Gamma_G[v]$. If no such neighbor exists then $v$ is in $G_P$. This involves a single communication round where $v$ sends the list of its neighbors to all its neighbors. In one additional round, each node $v$ informs its neighbors whether $v \in G_P$.
2. **Compute a distance-$k$ maximal independent set $J$ of $G_P$:** The nodes simulate MisBI on graph $G_P^k$, where nodes are adjacent if they are at distance at most $k$ in $G_P$, in $O(k \cdot \log^* n)$ rounds. The result is a distance-$k$ maximal independent set $J$ of $G_P$.
3. **Color inclusive neighborhoods of $J$:** Each dominator $v \in J$ colors its inclusive neighborhood $\Gamma_G[v]$ optimally using at most $\chi(G)$ colors.
4. **Color remaining nodes:** For any two dominators $u, v$ with $dist(u, v) < 2k$ and $ID(u) > ID(v)$, $u$ colors all uncolored nodes between $u$ and $v$ in $O(k)$ rounds as follows: $u$ collects its $2k$-neighborhood including the color constraints given by the already colored neighborhood of $v$. The best coloring of the remaining nodes is computed locally and newly colored nodes are informed of their color.

The key part of our analysis is to show that Step 4 of the algorithm does not require too many colors. To this end, we employ a result by Valencia-Pabon [22] on coloring *circular-arc graphs*, which are the intersection graphs of a set of arcs on a circle. Given a circular arc graph $F$, the *load* $L(F)$ is the cardinality of the largest subset of arcs containing the same point. The *circular-cover* $l(F)$ is the cardinality of the smallest subset of arcs that cover the entire circle. See Fig. 1 for an example.

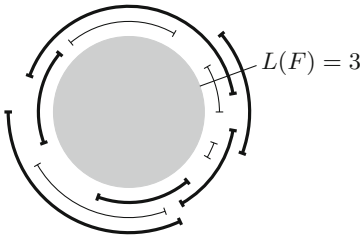



**Fig. 1.** A 3-colorable circular arc graph $F$ with load $L(F) = 3$ and circular-cover $l(F) = 6$. The vertices of a circular-cover are illustrated in bold. The result of Valencia-Pabon (Theorem 1) gives an upper bound of four colors.

**Theorem 1 (Valencia-Pabon [22]).** *Let $F$ be a circular arc graph with load $L$ and circular-cover $l \geq 4$. Then $\lfloor \left(1 + \frac{1}{l-2}\right) L \rfloor + 1$ colors suffice to color $F$.*

Equipped with Theorem 1, we prove now the existence of a good coloring that is required in Step 4 of our algorithm.

**Lemma 1.** *Let $G = (V, E)$ be an interval graph, $C_1, C_2 \subseteq V$ disjoint maximal cliques such that $dist(v_1, v_2) \geq l$ for every pair of nodes $v_1 \in C_1, v_2 \in C_2$, for an integer $l \geq 5$. Let $G' = G[C_1 \cup C_2 \cup D]$, where $D \subseteq V$ is the set of intervals located between $C_1$ and $C_2$. Let $\gamma$ be a partial coloring of $G'$ such that $\gamma(v) \in [\chi(G')]$ for every $v \in C_1 \cup C_2$, and $\gamma(v) = \perp$ for $v \in D$. Then, $\gamma$ can be extended to a coloring that employs at most $\lfloor (1 + \frac{1}{l-3})\chi(G') \rfloor + 1$ colors.*

*Proof.* Let $F$ be the graph obtained from $G'$ by contracting every pair of vertices $v_1 \in C_1$, $v_2 \in C_2$ with $\gamma(v_1) = \gamma(v_2)$. We will argue that $F$ is a circular arc graph with load $\chi(G')$ and circular-cover $l-1$. Our result then follows from Theorem 1.

A representation of $F$ with circular arcs can be obtained by, first, wrapping the line segment that contains all intervals of $G'$ onto an arc $A \subsetneq C$ of a circle $C$, and then replacing every pair of intervals/arcs $v_1 \in C_1, v_2 \in C_2$ with $\gamma(v_1) = \gamma(v_2)$ with an arc of minimal length that includes $v_1$, $v_2$ and all points of $C \setminus A$. See Fig. 2 for an illustration.

Since we replaced at most $\chi(G')$ pairs of arcs with arcs that cover $C \setminus A$, every point of the arc $C \setminus A$ is covered by at most $\chi(G')$ arcs. Furthermore, since all points on the arc $A$ are covered by at most $\chi(G')$ arcs, we obtain $L(F) = \chi(G')$. The circular-cover is at least the length of the shortest path in $G'$ from $C_1$ to $C_2$ minus one, i.e., $l - 1$, since all arcs of $D$ need to be covered, and pairs of nodes of $C_1$ and $C_2$ are contracted in $F$. □

Using Lemma 1, we show next that our algorithm gives a $(1 + \epsilon)$-approximation guarantee:

**Theorem 2.** *Let $G = (V, E)$ be an interval graph and suppose that $\epsilon \geq \frac{2}{\chi(G)}$. Then, there is a deterministic $(1+\epsilon)$-approximation algorithm for coloring interval graphs in the $\mathcal{LOCAL}$ model that runs in $O(\frac{1}{\epsilon} \log^* n)$ rounds.*

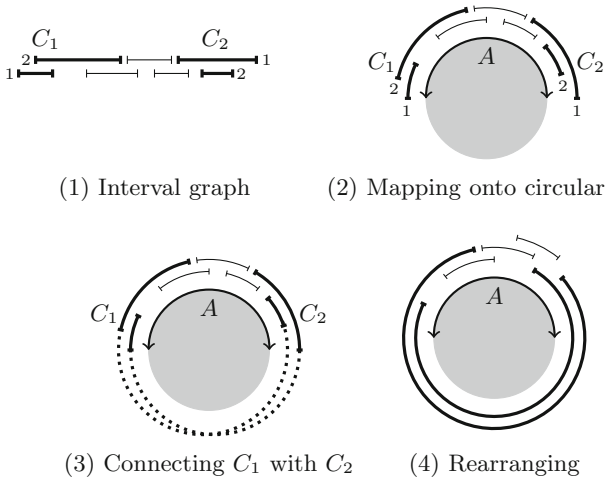



**Fig. 2.** Construction used in the proof of Lemma 1. Edges of the maximal cliques $C_1$ and $C_2$ are illustrated in bold. The colors of the intervals in $C_1$ and $C_2$ are indicated by the small numbers next to the intervals. In this example, a path of length 7 is mapped onto an arc $A$. Then pairs of vertices with the same color of the maximal cliques $C_1$ and $C_2$ are connected, which gives a cycle of length 5. Every coloring of $C_5$ requires 3 colors.

*Proof.* Let $k = \lceil \frac{2}{\epsilon} \rceil + 5$. Our algorithm computes a distance-$k$ maximal independent set $J$ in the subgraph of proper intervals. Observe that $J$ is also a distance-$k$ independent set in $G$, since for every shortest path in $G$ between two nodes of $V_P$, there is one that only traverses edges of $G_P$. Let $u, v$ be two nodes as in step 4 of the algorithm such that $u$ is left of $v$. Let $C_1$ be the set of intervals that intersect $u$'s right boundary, and let $C_2$ be the set of intervals that intersect $v$'s left boundary. Then, $C_1$ and $C_2$ are maximal cliques and were colored in step 3 of the algorithm. Since $J$ is a distance-$k$ independent set, every pair of nodes of $C_1$ and $C_2$ are at a distance of at least $k-2$. Then, by Lemma 1, the coloring can be completed using $\chi(G)(1 + \frac{1}{(k-2)-3}) + 1 = \chi(G)(1 + 1/\lceil \frac{2}{\epsilon} \rceil) + 1$ colors, which simplifies to:

$$\chi(G)(1 + \frac{1}{\lceil \frac{2}{\epsilon} \rceil}) + 1 \leq \chi(G)(1 + \frac{\epsilon}{2}) + \frac{\epsilon\chi(G)}{2} \leq (1+\epsilon)\chi(G),$$

where we used the assumption $\epsilon > \frac{2}{\chi(G)}$ which implies $1 \leq \frac{\chi(G)\epsilon}{2}$. The runtime of the algorithm is dominated by the computation of the distance-$k$ maximal independent set in step 2, which runs in $O(k \cdot \log^* n) = O(\frac{1}{\epsilon} \log^* n)$ rounds. □

### 3.2 Adapting the Algorithm to the $\mathcal{CONGEST}$ Model

We now adapt the previous algorithm to the $\mathcal{CONGEST}$ model when each node $v_i \in V$ knows its interval representation boundaries $a_i, b_i$. We assume that representing the numbers $a_i, b_i$ uses logarithmic space.

We reuse Steps 1 and 3 from our previous algorithm. Step 2, i.e., finding a distance-$k$ maximal independent set in $G_P$ by running MISBI on $G_P^k$ cannot be implemented in the $\mathcal{CONGEST}$ model, since the nodes cannot collect their full distance-$k$ neighborhoods quickly. Instead, we replace this step by a subroutine that finds a $(k, 3k/2+1)$-*ruling set*. A $(p, q)$-ruling set in graph $G = (V, E)$ is a subset of vertices $I \subseteq V$ such that every pair of vertices in $I$ are at distance at least $p$, while every vertex outside $I$ is at distance at most $q$ from a vertex in $I$. We will argue that such a set can be computed in the $\mathcal{CONGEST}$ model in $O(k \log^* n)$ rounds on graph $G_P$. Step 4 is replaced by a more technical coloring process.

Regarding Step 1, exchanging interval boundaries is enough in order to determine whether a node $v \in V$ is also in $V_P$. For a node $v \in J$ to color its neighborhood optimally in Step 3, it requires only the interval representation of its neighboring nodes in order to determine the neighborhood relations among them. This information can be exchanged in one round in the $\mathcal{CONGEST}$ model.

**Computing a $(k, 3k/2+1)$-ruling set in $G_P$.** We next argue how to compute a $(k, 3k/2 + 1)$-ruling set in $G_P$, the subgraph of proper intervals. First, we compute a maximal independent set $I_1$ in $G_P$ using MISBI. We then proceed inductively. Let $v_1, v_2, \ldots$ denote the intervals of $I_j$ ordered from left to right. We build an auxiliary graph $G_j$ on vertex set $I_j$, where node $v_i$ is adjacent to nodes $v_{i-1}$ and $v_{i+1}$. Notice that $G_j$ is an interval graph (it is in fact a path). We then compute a maximal independent set $I_{j+1}$ in graph $G_j$ using MISBI (or for example the Cole-Vishkin algorithm [9]).

We now argue that $I_{\lceil \log k \rceil}$ is a $(k, 3k/2+1)$-ruling set in $G_P$, and the computation of $I_k$ can be implemented in the $\mathcal{CONGEST}$ model in $O(k \log^* n)$ rounds. Let $v_1, v_2, \ldots$ denote the intervals of $I_j$ ordered from left to right. Let $l_j$ and $u_j$ denote the minimum and maximum distance between vertices in $I_j$, respectively. Since $I_1$ is a maximal independent set, we have $l_1 \geq 2$ and $u_1 \leq 3$. Then, it is easy to see that $l_{j+1} \geq 2l_j \geq 2^{j+1}$ and $u_{j+1} \leq 2u_j \leq 3 \cdot 2^j$. Thus, $l_{\lceil \log k \rceil} \geq k$ and $u_{\lceil \log k \rceil} \leq 3k$. Furthermore, every vertex outside $I_{\lceil \log k \rceil}$ is at distance at most $3k/2 + 1$ from a vertex of $I_{\lceil \log k \rceil}$.

Concerning the runtime, simulating the run of MISBI on $G_j$ increases the runtime of MISBI by a factor of $u_j$. Hence, computing $G_j$ requires $O(3 \cdot 2^{j-1} \log^* n) = O(2^j \log^* n)$ rounds. Overall the runtime for computing $I_{\lceil \log k \rceil}$ is $\sum_{j \leq \lceil \log k \rceil} O(2^j \log^* n) = O(k \log^* n)$ rounds.

**Coloring.** After Step 3 of the algorithm, we have computed a $(k,\ 3k/2 + 1)$-ruling set $J$ and a partial coloring $\gamma$ such that nodes $\cup_{w \in J} \Gamma_G[w]$ are colored while all other nodes are uncolored, i.e., $\gamma(z) \in [\chi(G)]$, if $z \in \cup_{w \in J} \Gamma_G[w]$, and $\gamma(z) = \perp$, otherwise. Every pair of adjacent nodes $u, v \in J$ with $u$ located left of $v$ executes the coloring procedure presented in the following in order to color the uncolored nodes located between them.

Fix two nodes $u, v \in J$ as described above. In the description of our algorithm, we use the following notations. Denote by $C_1$ ($C_2$) the maximal clique consisting of intervals that intersect $u$'s right boundary (resp. $v$'s left boundary). Let $D$ be

the set of intervals outside $C_1 \cup C_2$ located between $u$ and $v$. Let $N_i \subseteq C_1 \cup C_2 \cup D$ be the set of nodes at distance $i$ from $u$, and let $N_{\geq i} = \cup_{j \geq i} N_j$. We also ensure that every node $a \in D$ learns its distance from $u$ (and thus the index $i$ such that $a \in N_i$). This can be established by flooding the network with a token initially broadcasted by $u$, and the number of rounds it takes until the token reaches node $a \in D$ equals $dist_G(a, u)$. Denote by $n_i$ the interval of $N_i$ that reaches out furthest to the right. Nodes of $N_i$ can identify this interval easily by communicating their interval boundaries to their neighbors.

Our coloring procedure requires an implementation of the Greedy coloring algorithm as a subroutine in the $\mathcal{CONGEST}$ model.

**Greedy Coloring Subroutine.** W.l.o.g. we present a left-to-right coloring initiated by node $u$; a right-to-left coloring initiated by $v$ can be obtained similarly. First, node $n_1$ colors the uncolored nodes of its neighborhood: It traverses its uncolored neighbors with increasing left interval boundary and assigns the smallest possible color. Then, $n_2$ continues with the same process. The coloring carries on until all $n_i$ have colored their neighborhoods. The runtime of this procedure is $\mathrm{O}(dist_G(u, v)) = \mathrm{O}(k)$.

**Coloring Process.** Node $u$ initiates a left-to-right Greedy coloring $\gamma_1$ that respects the colors given by $\gamma$ of $C_1$ (and not necessarily the colors of $C_2$), and simultaneously, $v$ initiates a right-to-left Greedy coloring $\gamma_2$ that respects the colors given by $\gamma$ of $C_2$ (and not necessarily the colors of $C_1$). We then transform the coloring $\gamma_1$ into one that respects the colors $\gamma_2$ on $C_2$.

Our algorithm operates in $p$ phases, each consisting of three recoloring steps. In phase $i$, we alter the coloring $\gamma_1$ of nodes $N_{\geq 3i}$ such that $\gamma_1$ is non-conflicting with colors $T_i = \{(i-1)B + 1, \ldots, iB\}$ of $\gamma$ on $C_2$, where $B = \lceil \frac{\chi(G)}{p} \rceil$. To this end, nodes of $N_{\geq 3i}$ with a color of $T_i$ recolor themselves to new colors $[\chi(G) + 1, \chi(G) + B]$ by adding $\chi(G) - (i-1)B$ to their own color. Then, nodes of $N_{\geq 3i+1}$ with a target color (the color given by $\gamma_2$) in $T_i$ recolor themselves to their target color. Last, we initiate a Greedy recoloring sweep at node $n_{3i+2}$ that recolors all nodes of $N_{\geq 3i+2}$ with a current color $> iB$ to colors in $\{iB + 1, \ldots, \chi(G)\}$.

We prove correctness of this algorithm in the following lemma.

**Lemma 2.** *After phase $i$ of the previous coloring process, the following holds:*

1. $\forall w \in N_{\geq 3i+2} : \gamma_1(w) \in [\chi(G)]$,
2. $\forall w \in N_{\geq 3i+1} : \gamma_2(w) \leq iB \Rightarrow \gamma_1(w) = \gamma_2(w)$,
3. *$\gamma_1$ is legal.*

*Proof.* Before iteration one (i.e. $i = 0$), all three items are trivially true. The first recoloring step of phase $i$ assigns nodes of $N_{\geq 3i}$ with current color in $T_i$ a color larger than $\chi(G)$. Note that this leads to a legal coloring, since by Item 1, none of the nodes of $N_{3i-1}$ are colored with a color larger than $\chi(G)$. In the second recoloring step, nodes of $N_{\geq 3i+1}$ with target color in $T_i$ receive their target color (which gives Item 2). Again, $\gamma_1$ remains legal since after the first recoloring step, none of the nodes of $N_{3i}$ are colored with a color in $T_i$. In the third recoloring

step, the Greedy coloring algorithm is executed on the subgraph induced by nodes $V_i = \{v \in N_{\geq 3i+2} : \gamma_1(v) \geq iB+1\}$. We claim that the algorithm recolors $V_i$ with colors in $[\chi(G)]$. Indeed, first note that $\chi(G[V_i]) \leq \chi(G) - iB$, since $\gamma_2$ restricted to $V_i$ gives such a coloring. Next, since the Greedy coloring algorithm processes the intervals with increasing left boundary, all color constraints when coloring an interval $x$ are imposed by intervals that intersect $x$'s left boundary (note that for two intervals $x \in N_j$, $y \in N_{j+1}$ we always have $l(x) < l(y)$). Since there are at most $\omega(G[V_i]) - 1 = \chi(G[V_i]) - 1$ such intervals, there is always an available color for $x$ in $[\chi(G)]$, which proves Item 1. Since legality of $\gamma_1$ is preserved throughout the three recoloring steps, Item 1 follows. □

This gives the following theorem:

**Theorem 3.** *Let $G = (V, E)$ be an interval graph and suppose that $\epsilon \geq \frac{2}{\chi(G)}$. Then, there is a deterministic $(1+\epsilon)$-approximation algorithm for coloring interval graphs in the $\mathcal{CONGEST}$ model that runs in $O(\frac{1}{\epsilon} \log^* n)$ rounds.*

*Proof.* Correctness of the algorithm was established in Lemma 2. By construction, the algorithm uses at most $\chi(G)(1 + \frac{1}{p}) + 1$ colors. Thus, we set $p = \lceil \frac{2}{\epsilon} \rceil$ which implies that the number of colors is bounded by $(1+\epsilon)\chi(G)$, which proves the approximation guarantee. In order to execute $p$ phases of the color rotation algorithm, it is necessary that adjacent nodes of $J$ are far enough apart. To this end, we set parameter $k$ to $k = 3p + 2 = O(\frac{1}{\epsilon})$.

Concerning the runtime of the algorithm, besides an $O(k \log^* n)$ term for the computation of the $(k, 3k/2 + 1)$-ruling set, an additive $O(k^2)$ term is incurred by the Greedy coloring algorithm: In each of the $O(k)$ phases, we execute the Greedy coloring algorithm which requires $O(k)$ steps. We argue now that the $k^2$ term can be reduced to $k$ by pipelining the Greedy recoloring sweeps. Iteration $i$ can be started as soon as nodes $N_{3i+2}$ have been recolored by the recoloring sweep initiated in iteration $i - 1$. After the initiation of the recoloring sweep of iteration $i - 1$, it takes only a constant number of iterations until this sweep reaches nodes $N_{3i+2}$. Thus, iteration $i$ can be started after a constant number of iterations after the start of iteration $i-1$. Thus, by induction, iteration $k$ can be started $O(k)$ iterations after iteration 1 has been started. The overall runtime is thus $O(k \log^* n + k) = O(\frac{1}{\epsilon} \log^* n)$. □

*Remark:* In the recoloring step, we assume that nodes know $\chi(G)$. This can be circumvented as follows: The initial left-to-right coloring $\gamma_1$ is an optimal coloring of nodes $C_1 \cup C_2 \cup D$. Nodes in $C_1 \cup C_2 \cup D$ compute the largest color employed by $\gamma_1$. This value replaces $\chi(G)$ in the algorithm.

## 4 Lower Bound for Coloring Interval Graphs in the $\mathcal{LOCAL}$ Model

Linial's lower bound shows that every distributed algorithm for coloring the $n$-cycle with three colors requires time $\Omega(\log^* n)$. Since it is possible to decrement

the number of colors of a $c$-coloring, for $c \geq \Delta + 2$, in a single communication rounds using a standard method, Linial's lower bound even holds for coloring the ring with $O(\log^* n)$ colors. Furthermore, this lower bound can easily be adapted to hold for a path of length $n$ (which is also an interval graph and can be colored with two colors). it follows that computing a $O(\log^* n)$-approximate interval coloring requires $\Omega(\log^* n)$ rounds.

We present now a different lower bound argument that holds for interval graphs with arbitrary chromatic number. Specifically, we show that every distributed $(1+\epsilon)$-approximation algorithm for interval coloring requires $\Omega(\frac{1}{\epsilon})$ rounds.

To this end, we give a lower bound for multicoloring a path and provide a reduction between coloring intervals and path multicoloring.

Let $G(V, E, w)$ be a weighted path on $n$ vertices with $w(v) = k$, for every $v \in V$. Then the multichromatic number of $G$ is $\chi^m(G) = 2k$: alternate between the first $k$ and the second $k$ colors while traversing the path from left to right. We prove now that if $\phi$ is a $(1+\epsilon)$-approximate multicoloring of $G$, then the color sets of nodes at even distances have a large intersection and the color sets of nodes at odd distances have small intersection.

**Lemma 3.** *Let $\phi : V \to 2^{\mathbb{N}}$ be a $(1+\epsilon)$-approximate multicoloring of a path $G = (V, E, w)$ with $w(v) = k$, for every $v \in V$. Then, for $u, v \in V$ and an integer $r \geq 1$:*

1. *If $dist(u, v) = 2r$ then $|\phi(u) \cap \phi(v)| \geq k - 2kr\epsilon$,*
2. *If $dist(u, v) = 2r + 1$ then $|\phi(u) \cap \phi(v)| \leq 2kr\epsilon$.*

*Proof.* Since $\phi$ is a $(1+\epsilon)$-approximate multicoloring of $G$, we have $|\phi(V)| \leq 2(1+\epsilon)k$. Let $v_0, v_1, \ldots$ denote the vertices of the path so that $v_i$ and $v_{i+1}$ are adjacent. Then, by Item 1 of Definition 1, it holds that $|\phi(v_i) \cap \phi(v_{i+1})| = 0$. We further have $|\phi(v_i) \cap \phi(v_{i+2})| \geq k - 2\epsilon k$, since the total number of colors employed is bounded by

$$\begin{aligned} 2(1+\epsilon)k &\geq |\phi(v_{i+1})| + |\phi(v_i)| + |\phi(v_{i+2})| - |\phi(v_i) \cap \phi(v_{i+2})| \\ &= 3k - |\phi(v_i) \cap \phi(v_{i+2})|, \end{aligned}$$

which implies the claimed bound. Next, we use the relationship

$$\begin{aligned} |\phi(v_0) \cap \phi(v_{2r})| &\geq |\phi(v_0) \cap \phi(v_{2r-2})| - |\phi(v_{2r}) \setminus \phi(v_{2r-2})| \\ &= |\phi(v_0) \cap \phi(v_{2r-2})| - (k - |\phi(v_{2r}) \cap \phi(v_{2r-2})|) \\ &\geq |\phi(v_0) \cap \phi(v_{2r-2})| - 2\epsilon k, \end{aligned}$$

which implies $|\phi(v_0) \cap \phi(v_{2r})| \geq k(1-2r\epsilon)$ and proves Item 1. Last, since $|\phi(v_0) \cap \phi(v_1)| = 0$ and $|\phi(v_1) \cap \phi(v_{2r+1})| \geq k(1-2r\epsilon)$, we obtain

$$|\phi(u) \cap \phi(v_{2r+1})| \leq |\phi(v_{2r+1}) \setminus \phi(v_1)| \leq k - |\phi(v_1) \cap \phi(v_{2r+1})| = 2kr\epsilon,$$

which proves Item 2. □

Equipped with Lemma 3 we are ready to prove our lower bound on computing a multicoloring on the path. Let $G = (V, E, w)$ denote the path of length $n$ with $w(v) = k$, for every $v \in V$, and suppose that every vertex $v$ receives a unique label $\mathcal{L}(v)$, where $\mathcal{L}$ is chosen uniformly at random from the set of bijections between $V$ and $\{1, \ldots, n\}$.

**Theorem 4.** *Every possibly randomized distributed algorithm with error probability at most* $1/3$ *that computes a* $(1+\epsilon)$*-approximate multicoloring on a path* $G = (V, E, w)$ *with vertex weights* $w(v) = k$*, for every* $v \in V$*, requires at least* $\frac{1}{4\epsilon} - \frac{1}{2}$ *rounds.*

*Proof.* Let $v_1, \ldots, v_n$ denote the vertices of $G$ such that $v_i$ and $v_{i+1}$ are adjacent, and let $\phi$ denote the output multicoloring of the algorithm. Suppose that the algorithm runs in $r$ rounds. Consider an index $j$ such that $r+1 \leq j \leq n-3r-2$. Then, since the error probability of the algorithm is at most $1/3$ and by applying Lemma 3, we obtain

$$\mathbb{P}\left[|\phi(v_j) \cap \phi(v_{j+2r+2})| \geq k - 2k(r+1)\epsilon\right] \geq 2/3 \,, \tag{1}$$

$$\mathbb{P}\left[|\phi(v_j) \cap \phi(v_{j+2r+1})| \leq 2kr\epsilon\right] \geq 2/3 \,, \tag{2}$$

where the probabilities are taken over the coin flips of the nodes and the labeling function $\mathcal{L}$. Since the outputs of two nodes at distance at least $2r+1$ are independent (the output of a node is a function of the labels and random coin flips in its $r$-neighborhood), for every integer $c$ we obtain $\mathbb{P}\left[|\phi(v_j) \cap \phi(v_{j+2r+2})| = c\right] = \mathbb{P}\left[|\phi(v_j) \cap \phi(v_{j+2r+1})| = c\right]$ . Thus, Inequality 1 implies

$$\mathbb{P}\left[|\phi(v_j) \cap \phi(v_{j+2r+1})| \geq k - 2k(r+1)\epsilon\right] \geq 2/3 \,. \tag{3}$$

Suppose now that $r < \frac{1}{4\epsilon} - \frac{1}{2}$. Then, Inequality 3 gives

$$\mathbb{P}\left[|\phi(v_j) \cap \phi(v_{j+2r+1})| > k(\frac{1}{2} + \epsilon)\right] \geq 2/3 \,,$$

while Inequality 2 gives $\mathbb{P}\left[|\phi(v_j) \cap \phi(v_{j+2r+1})| < k(\frac{1}{2} + \epsilon)\right] \geq 2/3$, a contradiction. Thus, $r \geq \frac{1}{4\epsilon} - \frac{1}{2}$ holds which completes the proof. □

Finally, we provide a reduction from multicoloring the path to interval coloring.

**Theorem 5.** *Every possibly randomized distributed* $(1+\epsilon)$*-approximation algorithm with error probability* $1/3$ *for coloring interval graphs requires* $\Omega(\frac{1}{\epsilon} + \log^* n)$ *rounds.*

*Proof.* The $\Omega(\log^* n)$ part of the lower bound follows from Linial's ring coloring lower bound [16] (see also [12]).

To obtain the $\Omega(\frac{1}{\epsilon})$ part of the lower bound, consider an algorithm **A** as described in the statement of the theorem. Then, **A** can be used to compute a multicoloring of the path $G = (V, E, w)$ with $w(v) = k$, for any integer $k$, as follows: The nodes $v \in V$ simulate a *fat path* $G' = (V', E')$ where every vertex $v \in V$ is replaced by a clique $C(v)$ of size $k$ and two cliques $C(v)$ and $C(u)$ are adjacent if and only if $u$ and $v$ are adjacent in $G$. Such a fat path constitutes an interval graph and thus algorithm **A** can be simulated to compute a $(1+\epsilon)$-approximate coloring $\gamma$. We then set $\phi(v) = \cup_{v' \in C(v)} \gamma(v')$ which gives a $(1+\epsilon)$-approximation to the multicoloring problem. The simulation can be implemented with a constant factor blow-up on the number of communication rounds. The lower bound of Theorem 4 thus translates within a constant factor. □

## 5 Conclusion

In this paper, we have presented a distributed $(1+\epsilon)$-approximation algorithm for coloring interval graphs, which runs in $O(\frac{1}{\epsilon} \log^* n)$ rounds. It runs in the $\mathcal{LOCAL}$ model and can also be implemented in the $\mathcal{CONGEST}$ model if nodes are aware of their interval representations. We also gave a lower bound of $\Omega(\frac{1}{\epsilon})$. We further demonstrated that the color rotation technique employed in our $\mathcal{CONGEST}$ model algorithm can be useful for other coloring problems as well.

How can we extend our results to more general graph classes such as chordal graphs, which are a superclass of interval graphs? Since chordal graphs contain trees, Linial's lower bound on coloring trees [16] shows that every constant factor approximation algorithm for coloring chordal graphs requires $\Omega(\log n)$ rounds. Can we obtain a $(1+\epsilon)$-approximation on chordal graphs using $O(\text{poly}(\frac{1}{\epsilon}) \cdot \text{polylog}\, n)$ rounds in the $\mathcal{LOCAL}$ model? If nodes are aware of their index in a perfect elimination ordering of the chordal graph, can we obtain for example a $O(\text{poly}(\frac{1}{\epsilon}) \cdot \log^* n)$ rounds algorithm?

## References

1. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. J. Algorithms **7**(4), 567–583 (1986)
2. Barenboim, L.: Deterministic $(\Delta + 1)$-coloring in sublinear (in $\Delta$) time in static, dynamic, and faulty networks. J. ACM **63**(5), 47:1–47:22 (2016). http://dl.acm.org/citation.cfm?id=2979675
3. Barenboim, L., Elkin, M.: Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. Distrib. Comput. **22**(5), 363–379 (2010)
4. Barenboim, L., Elkin, M.: Distributed graph coloring: fundamentals and recent developments. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2013). http://dx.doi.org/10.2200/S00520ED1V01Y201307DCT011

5. Barenboim, L., Elkin, M., Gavoille, C.: A fast network-decomposition algorithm and its applications to constant-time distributed computation. In: Scheideler, C. (ed.) Structural Information and Communication Complexity. LNCS, vol. 9439, pp. 209–223. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25258-2_15
6. Barenboim, L., Elkin, M., Pettie, S., Schneider, J.: The locality of distributed symmetry breaking. J. ACM **63**(3), 20:1–20:45 (2016). https://doi.org/10.1145/2903137
7. Bollobás, B.: Chromatic number, girth and maximal degree. Discrete Math. **24**(3), 311–314 (1978)
8. Chang, Y.J., Kopelowitz, T., Pettie, S.: An exponential separation between randomized and deterministic complexity in the local model. In: Proceedings 57th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 615–624 (2016)
9. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Inf. Control **70**(1), 32–53 (1986). https://doi.org/10.1016/S0019-9958(86)80023-7
10. Fraigniaud, P., Heinrich, M., Kosowski, A.: Local conflict coloring. In: IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9–11 October 2016. Hyatt Regency, New Brunswick, New Jersey, USA, pp. 625–634 (2016). http://dx.doi.org/10.1109/FOCS.2016.73
11. Goldberg, A.V., Plotkin, S.A., Shannon, G.E.: Parallel symmetry-breaking in sparse graphs. SIAM J. Discrete Math. **1**(4), 434–446 (1988). https://doi.org/10.1137/0401044
12. Halldórsson, M.M., Konrad, C.: Distributed algorithms for coloring interval graphs. In: Kuhn, F. (ed.) DISC 2014. LNCS, vol. 8784, pp. 454–468. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45174-8_31
13. Harris, D.G., Schneider, J., Su, H.H.: Distributed (&#1)-coloring in sublogarithmic rounds. In: Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing, STOC 2016, pp. 465–478. ACM, New York (2016). http://doi.acm.org/10.1145/2897518.2897533
14. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, NewYork (1972)
15. Kuhn, F.: Local multicoloring algorithms: computing a nearly-optimal TDMA schedule in constant time. In: 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, 26–28 February 2009, Freiburg, Germany, Proceedings, pp. 613–624 (2009). http://dx.doi.org/10.4230/LIPIcs.STACS.2009.1852
16. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992). https://doi.org/10.1137/0221015
17. Luby, M.: A simple parallel algorithm for the maximal independent set problem. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC 1985, pp. 1–10. ACM, New York (1985). http://doi.acm.org/10.1145/22145.22146
18. Schneider, J., Elkin, M., Wattenhofer, R.: Symmetry breaking depending on the chromatic number or the neighborhood growth. Theor. Comput. Sci. **509**, 40–50 (2013). https://doi.org/10.1016/j.tcs.2012.09.004
19. Schneider, J., Wattenhofer, R.: An optimal maximal independent set algorithm for bounded-independence graphs. Distrib. Comput. **22**, 349–361 (2010)

20. Schneider, J., Wattenhofer, R.: A new technique for distributed symmetry breaking. In: Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2010, pp. 257–266. ACM, New York (2010). http://doi.acm.org/10.1145/1835698.1835760
21. Smith, D.A.: The first-fit algorithm uses many colors on some interval graphs. Ph.D. thesis, Arizona State University, Tempe, AZ, USA (2010). aAI3428197
22. Valencia-Pabon, M.: Revisiting Tucker's algorithm to color circular arc graphs. SIAM J. Comput. **32**(4), 1067–1072 (2003)
23. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput. **3**(1), 103–128 (2007)

# How Long It Takes for an Ordinary Node with an Ordinary ID to Output?

Laurent Feuilloley(✉)

Institut de Recherche en Informatique Fondamentale (IRIF),
CNRS and University Paris Diderot, Paris, France
feuilloley@irif.fr

**Abstract.** In the context of distributed synchronous computing, processors perform in rounds, and the time complexity of a distributed algorithm is classically defined as the number of rounds before all computing nodes have output. Hence, this complexity measure captures the running time of the slowest node(s). In this paper, we are interested in the running time of the ordinary nodes, to be compared with the running time of the slowest nodes. The *node-averaged* time-complexity of a distributed algorithm on a given instance is defined as the average, taken over every node of the instance, of the number of rounds before that node output. We compare the node-averaged time-complexity with the classical one in the standard LOCAL model for distributed network computing. We show that there can be an exponential gap between the node-averaged time-complexity and the classical time-complexity, as witnessed by, e.g., leader election. Our first main result is a positive one, stating that, in fact, the two time-complexities behave the same for a large class of problems on very sparse graphs. In particular, we show that, for LCL problems on cycles, the node-averaged time complexity is of the same order of magnitude as the "slowest node" time-complexity. In addition, in the LOCAL model, the time-complexity is computed as a worst case over all possible identity assignments to the nodes of the network. In this paper, we also investigate the *ID-averaged* time-complexity, when the number of rounds is averaged over all possible identity assignments of size $O(\log n)$. Our second main result is that the ID-averaged time-complexity is essentially the same as the expected time-complexity of *randomized* algorithms (where the expectation is taken over all possible random bits used by the nodes, and the number of rounds is measured for the worst-case identity assignment). Finally, we study the node-averaged ID-averaged time-complexity. We show that 3-colouring the $n$-node ring requires $\Theta(\log^* n)$ rounds if the number of rounds is averaged over the nodes, or if the number of rounds is averaged over the identity assignments. In contrast, we show that 3-colouring the ring requires only $O(1)$ rounds if the number of rounds is averaged over the nodes, and over the identity assignments.

The author received additional support from ANR project DESCARTES, and Inria project GANG.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 263–282, 2017.
https://doi.org/10.1007/978-3-319-72050-0_16

## 1 Introduction

The LOCAL model [22] is a standard model of distributed network computing. In this model, the network is abstracted as a graph, and the nodes perform in rounds to solve some task. At each round, each node can send messages to its neighbours in the graph, receive messages and perform some computation. The complexity of an algorithm solving some task is measured by the number of rounds before the task is completed, which usually depends on the size of the network, that is, its number of nodes.

A classic assumption in the LOCAL model is that the nodes know the size of the network *a priori*. As a consequence, in many algorithms, each node can compute from the start how many rounds are needed to solve the task, and stops after that number of rounds. There have been efforts to remove such *a priori* knowledge of the parameters of the graph (*e.g.* the arboricity [3] and the maximum degree [19]). Quite recently a general technique, called *pruning algorithms*, has been developed to remove the assumption that nodes know the size $n$ of the network [15]. In other words, [15] provides a method to transform a non-uniform algorithm into a uniform algorithm. The basic idea is to guess the number of nodes and to apply a non-uniform algorithm with this guess. The output can be incorrect, as the algorithm is only certified to be correct when it is given the actual number of nodes in the graph. The technique consists in virtually removing from the graph the nodes that have correct outputs, and to repeat the previous procedure with a new guess that is twice as large as the previous guess. Eventually all nodes have an output after a certain number of iterations, and the solution that is computed is correct. Note that with the resulting uniform algorithm some nodes can output very quickly, and some others can output much later. So far, only the classic measure of complexity, *i.e.* the time before all nodes stop and output, has been studied, even for such algorithms. In other words, only the behaviour of the *slowest* node has been considered. In this paper, we introduce a new measure of complexity, which is the *average* measure, in opposition to the usual measure which is a *worst-case* measure. More precisely, we define the running time of a node as the number of rounds before it outputs, and consider the average of the running times of the nodes. We argue that, when studying the locality of problems and of algorithms, it is worth to also consider this measure. Indeed it describes the typical local behaviour of the algorithm, that is, the behaviour of an *ordinary* node.

In some contexts partial solutions are useful. For example, consider the scenario in which two tasks are to be performed one after the other. In such case, it may happen that, on some part of the graph a partial solution for the first task is computed quickly. We can take advantage of this to start the second task in that part of the network, while the other nodes are still working on the first task. Note that knowing if the first task is finished can be impossible locally, and one has to design the second algorithms such it start at different rounds on different nodes. Consider a second scenario in which a global operator has to take a decision based on the outcome of a local algorithm. In that case, a partial solution may also be sufficient. For example the operator can detect that the

network is in a bad state, and start immediately a recovery procedure without waiting for all nodes to finish. Such situations are a motivation for the study of graph property testing, where a centralized algorithm probes the network on a sublinear number of nodes and take a decision based on this partial knowledge. We refer to the survey on graph property testing [13] for more examples of applications. When such partial solutions are useful, one would like to design algorithm that stop as soon as possible, and the average of the running time of the nodes is then a measure one would like to minimize.

Another classical assumption in the LOCAL model is that the nodes are given distinct identifiers. These identifiers (or IDs for short), are distinct binary strings on $O(\log n)$ bits, that is, distinct integers from a polynomially large space. The usual way to measure the complexity of an algorithm is again to consider its the worst-case behaviour, that is, the performance of the algorithm on the worst ID assignment. We argue that the average performances over all ID assignments, is also worth considering. Indeed many lower bounds are based on the fact that, as the identifiers can be viewed as set by an adversary, they do not really help to break symmetry. For example, on a path, one may consider the identifier assignment $1, 2, ..., n$, and argue that if nodes only consider the relative ordering of the identifiers in their neighbourhoods, then many nodes have the same view, and thus they cannot break symmetry. It is interesting to study if such specific constructions are required, or if one can design lower bounds that are robust against arbitrary ID assignment. We cannot expect that IDs are always set in a perfect way for the task we consider, but it may seem excessive to consider that they are set in an adversarial way, which naturally leads to the question of random assignments. We study the complexity of algorithms on random ID assignment, as the average over all possible ID assignment of the running time of the slowest node. Finally, the typical behaviour of an algorithm can arguably be the expected running time of an ordinary node on a random ID assignment. That is, the standard complexity but averaged on both nodes and ID assignments.

For the sake of concreteness, here is an example of the type of questions tackled in this paper. Consider the classic task of 3-colouring a ring of $n$ nodes. It is known that this task requires $\Omega(\log^* n)$ rounds [16]. This bound also holds for randomized algorithms [20]. The question tackled in this paper are of the following form: is it the case that a node typically outputs after a constant number of rounds, or is the $\Omega(\log^* n)$ lower bound robust to this spatial averaging? And what about the complexity of the problem on a random ID assignment?

*Our results.* Our first result is that averaging on the nodes can have a dramatic effect on the time complexity of solving a task in the LOCAL model. Indeed, for leader election on cycles, there is an exponential gap between the node-averaged complexity and the classic complexity. That is the slowest node outputs after a number of rounds that is exponentially larger than the time complexity of an ordinary node. This contrasts with our next result, for very sparse graphs. We say that a graph is $q$-sparse, if every ball of radius $r$ has at most $q.r$ nodes. For $q$-sparse graphs, we show that, for many classic tasks, the two measures are of the same order of magnitude. More precisely for a class of tasks that generalizes

the class of *locally checkable labellings* (LCL for short) [21], we show the following lemma, that we call *local average lemma*. For a given algorithm, either no node has a running time much larger than the average in its neighbourhood, or there exists an algorithm that is strictly better, that is an algorithm that has smaller running time for every node in every graph. As a consequence when proving lower bounds for these problems, one can use the fact that there is no *peak* in the distribution of the running times of the nodes. Then, to show that the average running time is large, it suffices to show that there is a large enough number of nodes that are far enough one from the other and that have large running time . This local average lemma can be used to show, for example, that for LCL problems on cycles, the landscape of complexities for an ordinary node and for the slowest node is the same, that is, for every problem the complexity is either $\Theta(1)$, $\Theta(\log^* n)$ or $\Theta(n)$.

We then move on to averaging on the identifier assignments. That is, we consider the expected behaviour of deterministic algorithms on random ID assignments. This topic happens to be related with the expected complexity of randomized algorithms. We show that even though these two models have specific properties, namely the independence of the random strings for the randomized algorithms, and the uniqueness of the identifiers for random ID assignment, the complexities are essentially the same. It follows that the results known for randomized algorithms can be translated to average over the identifiers.

Finally we prove that averaging on both nodes and IDs, can have an important effect on the complexity. We take the example of 3-colouring an $n$-node cycle. From the previous results of the paper, and from the literature, we know that this task has complexity $\Omega(\log^* n)$ for both the average on the nodes and the average on the identifiers. Quite surprisingly, when averaging on both the nodes and the ID assignment, the complexity becomes constant. In other words, deterministic and randomized complexity of ordinary nodes are clearly separated. Such separation contrast with the situation when considering the classic measure, as randomized constant-time algorithms for LCL, can be derandomized to get constant-time deterministic algorithms [21].

*Related works.* The LOCAL model was defined in [16], and a standard book on the topic is [22]. The problem of leader election, studied in Sect. 3, is a classic problem in distributed computing [2,18].

Deterministic algorithms stopping after different number of rounds on different nodes have been studied in contexts where the parameters of graphs, such as the degree or the number of vertices, are unknown. Such algorithms are called uniform algorithm, because it is the same algorithms that is run on every graph, independently of the parameters. A work that is particularly relevant to us is [15]. In this paper the authors prove that for a wide class of problems, one can remove the assumption that the nodes know the size $n$ of the network. This is done by applying a general method to transform a non-uniform algorithm into a uniform algorithm, without increasing of the asymptotic running time. In this framework, called *pruning algorithms*, some nodes may stop very early and some

may run for much longer time. Such algorithms justify the study of the behaviour of an ordinary node and not only of the behaviour of the slowest node.

The local average lemma of Sect. 4 applies to problems that are local in the sense that the nodes can check in constant time if a given solution is correct. This is an extension of the well-studied notion of locally checkable labelling (or LCL for short) [21], which is similar but requires in addition that the size of the inputs and of the outputs are bounded. Also the set of correct labellings usually studied, *e.g.* in distributed decision [9], including in LCL, do not depend on the identifiers of the graph, a restriction that is not needed here.

Randomized algorithms that turn out to be equivalent to algorithms working on random ID assignment form a well-studied subject, going back to the 80s with algorithms for maximal independent sets [1,17]. Recently, improvements on classic problems have been obtained [12,14] along with an exponential separation between randomized and deterministic complexity [6] (see also [4]). In [12], the author, by advocating the study of the so-called *local complexity* for a randomized algorithms, conveys the same message as the current paper: the behaviour of a typical node is worth considering, even if some nodes of the graph have much worst behaviour.

In this paper, we consider two relaxations of the measure of complexity, from worst-case to average, on the nodes and on the IDs. An aspect that we do not consider is the structure of the graph. We refer to [11] and references therein, for the topic of local algorithms on random graphs.

Finally, part of the results of this paper appeared in a brief announcement at PODC 2015 by the current author [8].

## 2 Model and Definitions

The graph considered in this paper are simple connected graphs, and throughout the text $n$ will denote the number of nodes in the graph. The distance between two nodes is the number of edges on a shortest path between these nodes, that is, the hop-distance. The $k$-neighbourhood of a node $v$ in a graph $G$, is the graph induced by the nodes at distance at most $k$ from $v$. Every node is given a distinct identifier on $O(\log n)$ bits, or equivalently an integer from a polynomially large range.

The algorithms studied in this paper can be defined in two ways. In both definitions, the nodes are synchronized and work in rounds, and for both the computational power of the nodes is unbounded. In the first definition, at each round, every node can exchange messages with its neighbours, and perform some computation. There is no bound on the size of the messages. A given node chooses an output after some number of rounds, and different nodes can stop at different rounds. After the output, a node can continue to transmit messages and perform computations, but it cannot change its output. In other words, the nodes do not go to a sleep mode once they have output, but the output is irrevocable. In the second definition, each node starts with the information of its 0-neighbourhood, and increases the size of this view at each round. That is, after $k$ rounds, it

knows its $k$-neighbourhood, that is it knows the structure of the graph in this neighbourhood, along with the identifiers and the inputs of the nodes. At some round, it chooses an output and stops. These two definitions are equivalent. On one hand, if we start from the first definition, we can assume that each round every node sends to its neighbours all the information it has about the graph (remember that the message size is unbounded)[1]. Then after $k$ rounds, a node has gathered the information about its $k$-neighbourhood. On the other hand, given a $k$-neighbourhood, a node can simulate the run of the other nodes, and compute the messages that it would receive if the nodes were using a message-passing algorithm.

The running time of a node is the number of rounds before it outputs. With the second definition, the running time of the algorithm can be described in a more combinatorial way: it is the minimum $k$ such that the node can choose an output with a view of radius $k$. Given a graph $G$, an identifier assignment $I$ (from the set of legal ID assignments that we denote $\mathcal{ID}$), an algorithm $A$, and a node $v$, we denote by $r_{G,I,A}(v)$ the running time of node $v$ in this context. When the context is clear, we simply use $r(v)$. We now define the different measures of complexity used in this paper. Given a graph $G$, and an algorithm $A$, we call *complexity of the slowest node complexity* or *classical complexity*, and *complexity of an ordinary node* or *node-averaged complexity* respectively, the following quantities:

$$\max_{I\in\mathcal{ID}} \max_{v\in G} r_{G,I,A}(v) \qquad \text{and} \qquad \max_{I\in\mathcal{ID}} \frac{1}{n}\sum_{v\in G} r_{G,I,A}(v)$$

In the second part of this paper, we consider the average on the identifier assignments and the average on both the identifiers and the nodes, that is, the following measures:

$$\frac{1}{|\mathcal{ID}|}\sum_{I\in\mathcal{ID}} \left(\max_{v\in G} r_{G,I,A}(v)\right) \qquad \text{and} \qquad \frac{1}{|\mathcal{ID}|}\sum_{I\in\mathcal{ID}} \left(\frac{1}{n}\sum_{v\in G} r_{G,I,A}(v)\right)$$

The tasks or problems that we want to solve in a distributed manner, are formalized with the notion of *language*. A language $\mathcal{L}$ is a set of configurations of the form $(G, I, x, y)$, where $G$ is a graph, $I$ an identifier assignment, and $x$ and $y$ are functions from the nodes of the graph to a set of labels. We are interested in constructing these languages, which means that given a graph $G$, an ID assignment $I$ and inputs given by the function $x$, we want to compute locally a function $y$ such that $(G, I, x, y)$ is in the language $\mathcal{L}$. The languages considered are such that for every $(G, I, x)$, there exists a legal output $y$. Note that usually, the identifier assignment is not part of the language [9,10,21], but our results hold for this more general version.

[1] There is a subtlety here, which is that after $k$ rounds in the message-passing algorithm a node cannot know the edges that are between nodes at distance exactly $k$ from it. For the sake of simplicity, we consider the proper $k$-neighbourhoods, as it does not affect the asymptotic of the algorithms.

In Sect. 3, we use the most general option regarding the knowledge of $n$ by the nodes, we assume such knowledge for lower bounds, whereas for upper bounds we do not require it. For Sect. 4, we assume that nodes do not have the knowledge of $n$. For the randomized part we assume this knowledge for the sake of simplicity, and refer the reader to Subsect. 4.4 of [15] for a technique to remove such assumption for randomized algorithm.

Throughout the paper, the expression *with high probability* means with probability at least $1 - 1/n$.

## 3 Exponential Gap for a Global Language

The complexity of an ordinary node is bounded by the complexity of the slowest node by definition. In this section, we show that the gap between these two quantities can be exponential.

**Theorem 1.** *The gap between the averaged-node complexity and the classical complexity can be exponential.*

We illustrate this phenomenon on the classic problem of leader election. The language of leader election is the set of graphs, with no inputs and binary outputs, such that exactly one node has label 1, and the others have label 0. It does not depend on the ID assignment. It is folklore that this problem has classic complexity $\Theta(n)$.

**Proposition 1 (Folklore).** *Leader election on an $n$-node ring requires $\Theta(n)$ rounds (for the slowest node).*

We prove this statement for completeness. The complexity of leader election in various models is discussed in [2,18].

*Proof.* Let $A$ be an algorithm for leader election, which has access to the size of the graph. Suppose that the slowest node complexity of $A$ is $c(n) \in o(n)$. Let $n_0$ be a large enough constant such that $2c(n_0) + 1 < n_0/2$. Consider a ring $R_1$ of length $n_0$. After running the algorithm $A$ on $R_1$, a node $v_1$ is elected to be the leader. This node $v_1$ outputs 1, after at most $c(n_0)$ steps. That is, $v_1$ outputs based on a view that contains at most $2c(n_0)+1$ nodes. Because of the definition of $n_0$, this view contains less than $n_0/2$ nodes. Let $I_1$ be the set of identifiers in this view. Now consider another ring $R_2$ of length $n_0$, whose set of identifiers does not contain any of the IDs of $I_1$. Again, a node $v_2$ is designated as the leader, and its view contains less than $n_0/2$ nodes. Now consider the ring made by concatenating the two views, and adding dummy nodes with fresh identifiers, to make sure that the ring has size $n_0$. Because the identifiers are all distinct, this is a proper instance. Then, as $v_1$ and $v_2$ have the same view as in $R_1$ and $R_2$ respectively, with the same graph size $n_0$, they output the same as in $R_1$ and $R_2$. That is, they both output 1, and thus produce a configuration that is not in the language, which a contradiction. □

**Proposition 2.** *The complexity of an ordinary node for leader election on an $n$-node ring is $O(\log n)$.*

*Proof.* Consider the following algorithm. Each node increases its radius until one of the two following situation occurs. First, if it detects an ID that is larger than its own, then it outputs 0. Second, if it can see the whole ring and that no ID is larger than its own, then it outputs 1. It is easy to see that this algorithm is correct as only the node with maximum ID can output 1. Note that this algorithm is order-invariant in the sense of [21], *i.e.* the algorithm does not take into account the identifiers themselves, but only their relative ordering in its view. In particular, the algorithm does not require the knowledge of $n$. We show that the averaged-node complexity of this algorithm is logarithmic in $n$.

Let us first make an observation. Consider the nodes with the $k$ largest identifiers, and mark them. The nodes that are not marked form $k$ paths, some of them possibly empty. A key property is that the behaviour of the algorithm on one path is independent of the other paths. More precisely, we claim that on a given path the algorithm will have the same behaviour whatever the sizes and the identifier distributions of the other paths are. Fix a path, and a node $v$, in this path. By definition, $v$ has an identifier that is smaller than the ones of the two marked nodes at the endpoints of the path. Therefore, it stops either before, or just when reaching one of the marked nodes, and it outputs 0. As a consequence it will never get to know the rest of the cycle. This simple observation implies that we can study the behaviour of the algorithm on each path separately. Let $p$ be integer, and let us consider a path of length $p$ with two additional marked nodes at each endpoint. In order to study the behaviour of the algorithm on this path, it is sufficient to consider all the relative ordering of identifiers on this path, because it is an order-invariant algorithm. Marked nodes can be replaced by nodes with IDs larger than every ID in the path. Let $a(p)$ be the maximum over all these identifier assignments of the sum of the running time of the nodes. We claim that this function obeys the following recursion formula:

$$a(p) = \max_{1 \leq k \leq \lceil p/2 \rceil} \{k + a(k-1) + a(p-k)\}.$$

Consider the node $v$ with the largest identifier in the path. It must reach one of the endpoints to stop. Then if we mark this node, the behaviour of the algorithm on the two subpath is independent of the context, and the maximum sums of running times in each path is $a(p_1)$ and $a(p_2)$ for the first subpath of length $p_1$ and the second of length $p_2$ respectively. Then the only parameter is the distance $k$ from $v$ to the closest endpoint. Given such an integer $k$, $a(p)$ is then equal to $k + a(k-1) + a(p-k)$. One can then check by induction that this maximum is always met for the value $k = \lceil p/2 \rceil$. Then an alternative formula is:

$$a(p) = \left\lceil \frac{p}{2} \right\rceil + a\left(\left\lceil \frac{p}{2} \right\rceil\right) + a\left(\left\lceil \frac{p}{2} \right\rceil - 1\right)$$

The sequence $a(n)$, defined by the induction formula above, along with initial values $a(0) = 0$ and $a(1) = 1$, is known to be in $\theta(n \log n)$. For references and

more information about this sequence, see [25]. Consequently, the sum of the running times of the nodes is equal to the sum of the running time of the leader, which is $n/2$, and of $a(n-1)$. This is because, we can mark the node that has the largest ID, and consider the rest of graph as a path. Thereafter, the complexity of an ordinary node is logarithmic in $n$. □

Note that analysis of the same flavour already exist in the literature, see for example [24] p. 125. Theorem 1 follows from Propositions 1 and 2.

## 4 Local Average Lemma and Application

This section is devoted to proving that, for local languages on very sparse graphs, the complexity of an ordinary node is basically the same as the one of the slowest node. This proof is based on a *local average lemma*. Given a graph and an algorithm, let us define informally a *peak*, as a node with high running time, whose neighbours at some distance have much smaller running times in average. The lemma states that, for local languages, and for algorithm that are somehow optimal, there is no such peak.

In order to give an intuition of this lemma, let us use the example of the 3-colouring a cycle. Consider an algorithm for the problem, and three adjacent nodes $u$, $v$ and $w$, in this order, in a cycle. We claim that if $r(v) > \max(r(u), r(w)) + 1$, then the algorithm can be speeded up. Indeed after $\max(r(u), r(w)) + 1$ steps, $v$ can simulate the computation of $u$ and $w$, deduce the colours they output, and output a non-conflicting colour. As a consequence if one wants to prove a lower bound on the average of the running times, one can assume that, for every node, at least one of its neighbours has a similar running time, namely at least its running time minus one.

In this section the algorithm do not have the knowledge of $n$. In order to state the lemma we need to introduce a few notions.

*Class* $\mathsf{LCL}^*$. We consider a large class of distributed problems that we call $\mathsf{LCL}^*$, which includes the well-known class of $\mathsf{LCL}$ problems [21], and the more general class $\mathsf{LD}$ [10]. A language $\mathcal{L}$ is in $\mathsf{LCL}^*$, if there exists a constant-time *verification algorithm*. That is, an algorithm $\mathcal{V}$ performing in a constant number of rounds, with binary output, *accept* or *reject*, such that for every configuration $(G, I, x, y)$, $\mathcal{V}$ accepts at every node, if and only the graph is in the language $\mathcal{L}$. The running time of $\mathcal{V}$ is called the *verification* radius. No bound on the size of the inputs and output is necessary, and the language can depend on the identifiers.

*$q$-sparse graphs.* A graph is $q$-sparse if any ball of radius $r$ contains at most $qr$ nodes. For example a cycle is 3-sparse.

*Minimal algorithms.* The lemma has the following shape: given a node $v$ whose running time is $r$, the nodes of its neighbourhood have running times whose average is roughly $r$. This type of statement cannot hold in general as we could artificially increase the radius of a node by modifying the algorithm. But as we

are interested in lower bounds for the node-averaged complexity, we can consider algorithms that are in some sense minimal. More precisely, let $A$ and $A'$ be two distributed algorithms for some language $\mathcal{L}$. We say that $A$ is smaller than $A'$, if on every graph, every ID assignment and inputs, and on every node, the running time of $A$ is at most the running time of $A'$. For lower bounds on the node-averaged complexity, it is sufficient to study algorithms that are minimal for this ordering. Indeed, if an algorithm that is not minimal has low complexity, then there exists a minimal algorithms that has at most this complexity.

**Lemma 1 (Local average lemma).** *Let $\mathcal{L}$ be a language in $\mathsf{LCL}^*$ with verification radius $t$, and $A$ be a minimal algorithm for $\mathcal{L}$. There exists two positive constants $\alpha$ and $\beta$, such that on any $q$-sparse graph, ID assignment, inputs, and node $v$, the average of the running times of the nodes at distance at most $r(v)/2$ from $v$, is at least $\alpha.r(v) - \beta$.*

Let us denote by $B(v, k, G, I, x)$ the subgraph of $G$, with identifiers $I$, and inputs $x$, induced by the nodes at distance at most $k$ from a node $v$. Likewise, given two integers $k_1 < k_2$, let $S(v, k_1, k_2, G, I, x)$ be the induced graphs with IDs and inputs, induced by the set of nodes whose distance to $v$ is at least $k_1$ and at most $k_2$. When the context is unambiguous, we omit the information $G$, $I$ and $x$.

Let $\mathcal{L}$ be a language of $\mathsf{LCL}^*$. There exists a verification algorithm $\mathcal{V}$, such that a configuration $(G, I, x, y)$ is in the language $\mathcal{L}$ if and only if $\mathcal{V}$ accepts at all node. Let $t$ be the verification radius of $\mathcal{V}$. Let $\mathcal{L}$, $A$, $G$, $I$, $v$ and $x$ be respectively, a language, a minimal algorithm, a graph and an ID assignment, a node and an input assignment as in the lemma.

In order to prove the lemma, we will first prove the following claim.

*Claim.* For every integer $k$:

$$r(v) \leq 2k + 2t + \max_{u \in S(v,k,k+2t)} r(u)$$

*Proof.* Suppose the inequality does not hold for some $k$. Let us use the following notations:

$$M = \max_{u \in S(v,k,k+2t,G,I,x)} r(u) \qquad \text{and} \qquad B = B(v, k + 2t + M, G, I, x).$$

As in the example of 3-colouring at the beginning of this section, we define a new algorithm $A'$, designed to be smaller than $A$. On a node $w$ of a graph $G'$, with ID assignment $I'$, and inputs $x'$, the behaviour of $A'$ differs from the one of $A$ only if the following conditions are fulfilled:

(1) The running time of $w$, $r_{G',I',A}(w)$, is at least $2k + 2t + M$ ;
(2) The node $w$ is at distance at most $k$ from a node whose neighbourhood at distance $k + 2t + M$ is exactly $B(v, k + 2t + M, G, I, x)$.

See Fig. 1. When the two conditions are fulfilled, let $w_G$ be the node of $G$, whose position in $B$, ID, and input, are the same as the ones of $w$ in $G'$. In that

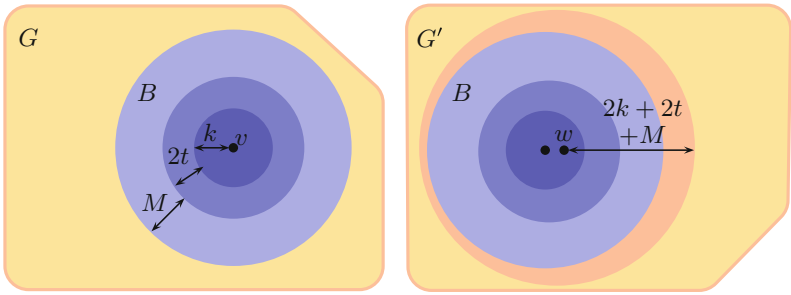



**Fig. 1.** This figure illustrates the definition of the algorithm $A'$ in the proof of Lemma 1. On the left is the original graph $G$ with node $v$, along with the ball $B$ around it. The behaviour algorithm $A'$ differs from the algorithm $A$ only if it is in the situation of the node $w$ on the right: it has running time at least $2k + 2t + M$, and it is at distance at most $k$ from a node whose $(k + 2t + M)$-neighbourhood is exactly $B$.

case, the algorithm $A'$ stops at round $2k + 2t + M$, and outputs the same label as $A$ does on $w_G$, in $(G, I, x)$.

The algorithm $A'$ is correct on $G$ by construction, as it has exactly the same outputs as $A$. We prove that $A'$ is correct on any graph. Consider the behaviour of the verification algorithm $\mathcal{V}$, on a node $z$ of a graph $G'$, with IDs $I'$ after the run of $A'$. This node may reject, only if it can detect a difference between the outputs of $A$ and $A'$. That is, only if $A'$ has an output that is different from the output of $A$, on a node $y$ of the $t$-neighbourhood of $z$. Because of the condition (2) in the definition of $A'$, $y$ is at distance at most $k$ from a node $v'$ whose neighbourhood at distance $k + 2t + M$ is exactly $B$. Then $z$ is at distance at most $t + k$ from this node $v'$. It is then sufficient to show that the algorithm $A'$ has correct outputs on the ball that is centred in $v'$ and has radius $k + 2t$, as the whole view of $\mathcal{V}$ on $z$ is contained in this ball.

First note that, the nodes at distance at most $k$ from $v'$ in $G'$, have the same output with $A'$, as the nodes at distance at most $k$ from $v$ in $G$ with $A$. This is by definition $A'$. Second, remember that $M = \max_{u \in S(v,k,k+2t,G,I)} r(u)$. As a consequence, in $G$, the nodes of $S(v, k, k + 2t, G, I)$ stop before they see nodes outside of $B$. The same holds in $G'$: as the $(k + 2t + M)$-neighbourhood of $v'$ is also $B$, the nodes of $S(v', k, k + 2t, G', I')$ have the same behaviour as is in the previous case, that is, they stop before they discover that they are not in $G$. Therefore, the nodes of $S(v', k, k + 2t, G', I')$ output as if they were in $S(v, k, k + 2t, G, I)$.[2] Then $A'$ has the same outputs as $A$, and as $A$ is correct, $A'$ is correct.

The algorithm $A'$ is strictly smaller that $A$, indeed no running time has been increased, and the running time of $v$ in $G$ has been reduced to $2k + 2t + M$. Consequently $A$ is not minimal, which is a contradiction. □

[2] Remember that the nodes do not have the knowledge of the size of the network, thus they have exactly the same information in $G$ and $G'$.

The final step of the proof Lemma 1, requires some computations, and to give more insights, we first give an intuition of this step, considering a simplified version of the inequality of Claim 4. Suppose we have the following inequality: $r(v) \leq \max_{S_k} r(u)$ where $S_k$ is the set of nodes at distance exactly $k$. The quantity $\max_{S_k} r(u)$ is upper bounded by $\sum_{S_k} r(u)$. Then, summing both terms of the inequality, for $k$ ranging from 1 to $r(v)$, one gets $r(v)^2 \leq \sum_S r(u)$, where $S$ is the ball of radius $r(v)$, without $v$. Now because of $q$-sparsity, there are at most $qr(v)$ nodes in $S$, and then $\sum_S r(u) \leq qa_S r(v)$, where $a_S$ is the average running time in $S$. Then $r(v) \leq q.a_S$.

We now finish the proof of the lemma. From Claim 4, for every $k$,

$$r(v) - 2k - 2t \leq \max_{u \in S(v,k,k+2t)} r(u).$$

The following inequality follows:

$$r(v) - 2k - 2t \leq \sum_{u \in S(v,k,k+2t)} r(u).$$

Let us sum the inequality above, for $k$ ranging from 1 to $r(v)/2 - 2t$. We assume without loss of generality that $t$ and $r(v)$ are positive. The left-hand term is then:

$$\sum_{k=1}^{r(v)/2-2t} (r(v) - 2k - 2t) = \frac{r(v)^2}{4} - tr(v) + \frac{r(v)}{2} - 2t \geq \frac{r(v)^2}{4} - 3tr(v).$$

The right-hand term is:

$$\sum_{k=1}^{r(v)/2-2t} \sum_{u \in S(v,k,k+2t)} r(u) \leq (2t+1) \times \sum_{u \in S(v,1,r(v)/2)} r(u).$$

Because of $q$-sparsity, the number of nodes in $S(v, 1, r(v)/2)$ is bounded by $(r(v)/2).q$, thus

$$(2t+1) \times \sum_{u \in S(v,1,r(v)/2)} r(u) \leq (2t+1) \frac{qr(v)}{2|S(v,1,r(v)/2)|} \sum_{u \in S(v,1,r(v)/2)} r(u).$$

Putting the pieces together, and simplifying the terms, we get:

$$\frac{r(v)^2}{4} - 3tr(v) \leq 2tqr(v) \frac{1}{|S(v,1,r(v)/2)|} \sum_{u \in S(v,1,r(v)/2)} r(u).$$

Dividing by $r(v)$, and defining $\alpha = \frac{1}{8tq}$ and $\beta = \frac{3t}{2q}$, we get:

$$\alpha.r(v) - \beta \leq \frac{1}{|S(v,1,r(v)/2)|} \sum_{u \in S(v,1,r(v)/2)} r_u$$

Which is the desired formula, as the right-hand term is the node-averaged complexity, and as $t$ and $q$ are constants. □

### 4.1 Applications

Thanks to the lemma, establishing a lower bound for node-averaged complexity of languages in $\mathsf{LCL}^*$ for very sparse graphs, boils down to show a simpler fact. It is sufficient to prove that a constant fraction of the nodes are close enough to nodes with running times similar to the running time of the slowest node. We illustrate this type of proof with $\mathsf{LCL}$ problems on cycles. It is known that for such problems, the slowest node complexity can only take three forms: $O(1)$, $\Theta(\log^* n)$ or $\Theta(n)$. See for example [5] for a recent presentation of this classification.[3] We prove that the landscape is the same for ordinary nodes.

**Theorem 2.** *For* $\mathsf{LCL}$ *on cycles, the node-averaged complexity has the same asymptotic classification as the slowest node complexity.*

*Proof.* Remember that the slowest node complexity is an upper bound on the average node complexity. Thereafter, it is sufficient to only prove the two lower bounds: $\Omega(\log^* n)$ and $\Omega(n)$.

Let us first focus on the case $\Theta(n)$. In this case, there exists a constant $\gamma$, such that on every cycle on $n$ nodes, for large enough $n$, at least one node $v$ has a running time at least $\gamma n$. As we consider a lower bound, we can assume that the algorithm is minimal. As Lemma 1 applies, the average complexity in the $(\gamma n/2)$-neighbourhood of $v$ is at least $\alpha\gamma n - \beta$, where $\alpha$ and $\beta$ are constants. Thereafter, the sum of the running time, in the $(\gamma n/2)$-neighbourhood of $v$ is bounded from below by $\alpha\gamma^2 n^2 - \beta\gamma n$ (or $n^2$). Hence the average complexity for the whole cycle is in $\Omega(n)$.

Now let us now consider the case of classical complexity $\Theta(\log^* n)$. Consider any minimal algorithm $A$ for the language $\mathcal{L}$ we consider. Again, let $\gamma$ be a constant, such that the slowest node complexity is at least $\gamma \log^* n$, for large enough $n$. There exists a ring $R_1$ on $n$ nodes, such that a node $v_1$ has running time $r_1 \geq \gamma \log^*(n)$. Then let $H_1$ be the graph that is composed of only the $r_1$-neighbourhood of $v_1$, and let $I_1$ be the set of identifiers of this segment. Now consider another ring $R_2$ on $n$ nodes, with no identifiers from $I_1$, such that there exists a node $v_2$ with running time $r_2 \geq \gamma \log^*(n)$. Let $H_2$ be $H_1$ concatenated with the $r_1$-neighbourhood of $v_1$. Note that because no identifier from $I_1$ is present in $R_2$, $H_2$ has distinct identifiers. This operation can be repeated, until $H_k$ has more than $n/2$ nodes. Let $H$ be $H_k$, completed in an arbitrary way to get a full cycle of size $n$ with distinct identifiers.

Note that as we performed the operation at most a linear number of times, the fact of removing some identifiers at each step is harmless as the identifier space is supposed to be polynomially large. Also note that the $\Theta(\log^* n)$ lower bound for the classical complexity is not affected by the constraints we add on the identifier space. This is because the lower bounds proofs do not rely on the particular shape of this space, which can even be assumed to be $\{1...n\}$[16].

[3] Even if not stated explicitly in [5], this classification also holds in the context where no knowledge of $n$ is assumed. This is because the $\Theta(\log^* n)$ bound relies on the construction of a maximal independent set, and that MIS is a problem for which the construction of [15] works.

We claim that on this cycle $H$ with this ID assignment, the average node complexity is $\delta \log^*(n)$ for some constant $\delta$. Indeed the nodes $v_i$, for $i$ ranging from 1 to $k$, have the same neighbourhood as in $R_i$ respectively, thus have running times $r_i$ respectively. Then using lemma 1 we get for every $i$ the nodes at distance at most $r_i/2$ from $v_i$ have running time at least $\alpha r_i - \beta$. And by construction there is a constant fraction of the nodes of $H$ that are in this case. As for every $i$, $r_i \geq \gamma \log^*(n)$, the node-averaged complexity is in $\Omega(\log^* n)$. □

This "extract and glue" technique works on other class of graphs, and similar bounds can thus be achieved. Nevertheless it is not true that, for any LCL problem and any graph class, the classic complexity is the same as the node-averaged complexity, as the following proposition shows.

**Proposition 3.** *There exists a graph class $\mathcal{C}$ for which 3-colouring has slowest node complexity $\Omega(\log^* n)$ but node-averaged complexity in $O(1)$.*

*Proof.* Consider first the following construction. Start with a path of even length $k$, and index the nodes along the path from $v_1$ to $v_k$. Create three new nodes and link them to the node $v_k$. Now for the nodes $v_i$ with $1 < i < k$, if the index $i$ is even, then add a node $v'_i$ and the edge $(v_i, v'_i)$. We call this construction a *short leg*. If the index $i$ is odd, add two nodes $v'_i$ and $v''_i$, and two edges $(v_i, v'_i)$ and $(v'_i, v''_i)$. This is a *long leg*. For both construction, the node $v_i$ is called the *basis* of the leg. Let us call such a graph an even-odd caterpillar. Now the graphs of $\mathcal{C}$ are the ones that can be constructed the following way: take an even-odd caterpillar based on a path of length $k$, and a cycle of length $\alpha \log^* k$ (where $\alpha$ is a large enough constant), and add an edge between an arbitrary node of the cycle and $v_1$. See Fig. 2.

Every algorithm must colour the $\alpha \log^* k$ cycle, and as the size of the graph is linear in $k$, the identifiers space is polynomial in $k$. Then Linial's lower bound applies on the cycle, and the slowest node complexity is $\Omega(\log^* k)$.

Let us now show that there exists an algorithm with constant node-averaged complexity for 3-colouring in this graph class. Every node first gathers its 3-hop neighbourhood. From this view it can deduce its position in the graph, and its behaviour for the following steps. More precisely, for every node $v$:

- if all the (direct) neighbours of $v$ have degree two, then it is a node of the cycle, then it runs the Cole-Vishkin procedure for 3-colouring a cycle [7]. It does not take into account the rest of the graph;
- if it is the basis of a short leg, or the middle of a long leg, then it takes colour 1;
- if it is the basis of a long leg, or has degree 1, then it takes colour 2;
- if it has degree four, then it is $v_k$ and it takes colour 1;
- if it has degree two and both its neighbours have degree three, then it is $v_1$, and it waits until both its neighbours have output, and it outputs a non-conflicting colour.

See Fig. 2.

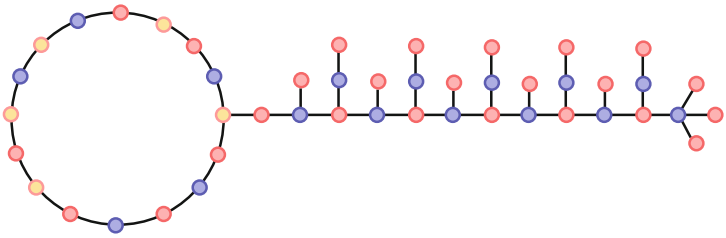


**Fig. 2.** The figure illustrates proof of Proposition 3. It takes $O(\log^* n)$ rounds to 3-colour the cycle on the left, but it take constant time to colour the even-odd caterpillar on the right, as a 2-colouring is hard-coded in the structure of the graph. In this picture, colour 1 is blue, colour 2 is red, and colour 3 is yellow. (Color figure online)

This algorithm uses at most $\log^* n$ rounds on the cycle and $v_1$, and constant time in the even-odd caterpillar. As the cyclic part has negligible size, the average node complexity is constant. □

## 5 Random ID Assignments and Randomized Algorithms

We move on to the second topic of this paper, where the randomized aspects are considered. The standard definition of the complexity in the LOCAL model not only considers the slowest node, but also the worst-case distribution of the identifiers. In this section we investigate the impact of replacing this measure by the running time of the slowest node, on a random ID assignments. In other words, given a graph, we consider the average of the slowest-node running time over all possible ID assignments.

The main result is the equivalence between such measure, and the complexity of randomized algorithms. Here, the complexity of a randomized algorithm is the expectancy of the number of rounds before every node finishes. Note that the two concepts have similar flavour, but are distinct On one hand, the random inputs of a randomized algorithm are independent, while in a random ID assignment, the identifiers are not independent. On the other hand, the IDs are distinct, while the random inputs can be equal. On a high level, the equivalence is similar to Yao's principle [26], that relates the performance of a randomized algorithm on a worst-case instance, and the complexity of a deterministic algorithm on a random instance. Also note that in the literature, the usual complexity of randomized algorithms is not the one we consider, but the time needed to output a correct solution with high probability. That is, Monte-Carlo algorithms are considered instead of Las Vegas algorithms. We discuss briefly this point at the end of the section.

For the following theorem, randomized algorithms are given random strings of size $O(\log n)$, and not infinite such strings. This hypothesis is not excessive as

most algorithm use a small amount of randomness. For example the celebrated MIS algorithm of [17] for bounded degree graphs, can be described as using random strings of size bounded by $O(\log n)$.

**Theorem 3.** *Given a problem, the expected slowest-node complexity of randomized algorithms, is equal to the expected deterministic slowest-node complexity on identifier assignment taken uniformly at random.*

*Proof.* It is part of the folklore that randomized algorithms do not need identifiers: they can generate such IDs with high probability by taking a integer in a cubic range uniformly at random. Here a probability of success equal to $1 - 1/n$ would be slightly too weak, so we make it $1 - 1/n^2$.

**Lemma 2.** *If $n$ numbers are taken independently uniformly at random between 1 and $n^4$, these numbers are pairwise distinct with probability $1 - 1/n^2$.*

*Proof.* The probability of two fixed numbers being equal is $1/n^4$. Then by union bound, the probability that a pair of numbers have the same value is bounded by the number of such pairs $n(n-1)/2$ multiplied by the former probability. Then the probability of collision is bounded by $1/n^2$, thus with probability $1 - 1/n^2$ the numbers are pairwise distinct. □

Remember that a randomized algorithm can be formalized as a deterministic algorithm having an auxiliary input, this input being a large enough random number. We consider an algorithm $A$ with an auxiliary input that can either be the ID or the random bits, and show that with high probability the behaviour is the same.

As stated in Proposition 2, taking independently and uniformly at random $n$ numbers from $[n^4]$ provides a list of distinct numbers with probability $1 - 1/n^2$. Also when this sampling succeeds, that is when the numbers are distinct, the outcome is uniform among all distinct identifiers assignments, because the identifiers are taken independently uniformly at random.

Let $D$ be a deterministic algorithm, and let $c$ be its average slowest-node complexity on identifier assignments taken uniformly at random. Let $R$ be a randomized algorithm, that first picks random numbers in $[n^4]$, and then runs $D$, until $D$ stops or until the node basically sees whole graph, and in the last case it outputs a colour such that the colouring is correct. The algorithm $R$ has probability at least $(1 - 1/n^2)$ to stop with $D$ that has expected runtime $c$, and probability at most $1/n^2$ to stop after at most $n$ rounds. Then the expected runtime is upper bounded by $(1 - 1/n^2)c + 1/n^2.n$ which asymptotically is $c$. Conversely, suppose that a randomized algorithm has expected complexity $c$. We claim that using the same algorithm using the identifier as random strings provides a deterministic algorithm with average complexity $c$. Suppose it is not the case. Then, the randomized algorithm must have complexity $c$ when the numbers are distinct, and $c'$ when they are non-distinct. The expected runtime is $(1 - 1/n^2)c + 1/n^2 c'$, which is asymptotically $c$ as $c'$ can be assumed to be at most $n$, which is a contradiction.

Thus Theorem 3 holds. □

A similar result can be obtained for the more classic context of Monte-Carlo algorithm. That is, when one considers the time before the nodes have stopped and output a proper solution with high probability, then the complexity of randomized algorithms and of deterministic algorithm on random identifiers are the same.

A related topic is to minimize the amount of randomness used by randomized algorithms. The amount of random bits necessary to perform a computation is usually not considered as a resource to be minimized in the LOCAL model. Whereas it is considered in centralized computing, see [23] for a precise example. Here, it is possible to do a small step in that direction, if we consider algorithms and languages that are local. In this case, it is not necessary to have all IDs of the graph that are different one from the other. In a local algorithm, the nodes see only a small neighbourhood of the graph, and thus only such neighbourhoods need to have distinct IDs. This is one of the ingredient of recent breakthroughs in the field, such as the speed-up theorem from [6] (see Theorem 6 in the paper).

Let $s$ be the maximum number of nodes that a node can see when it runs the local algorithm at hand. Then the following holds:

**Proposition 4.** *Taking uniformly at random numbers from* $\left[n^2 s^2\right]$ *is sufficient to have locally distinct identifiers with high probability.*

*Proof.* Consider a ball of size $s$. The probability that two nodes of this ball have the same identifier is upper bounded by $s^2/(n^2 s^2) = 1/n^2$. Then by union bound on all the centres of balls, one gets a probability of collision of $1/n$. □

### 5.1 Node-Averaged Randomized Complexity

After considering an average on the nodes, and on the identifiers assignment separately, we consider both averages together. That is we consider the behaviour of an ordinary node on an ordinary ID assignment. In the light of the previous subsection, this is equivalent to consider node-averaged complexity of randomized algorithms. This new measure can be unexpectedly low, as we illustrate on the example 3-colouring.

Theorem 2 implies that the node-averaged complexity of 3-colouring of a cycle is $\Theta(\log^* n)$. It is also known that the randomized complexity is $\Theta(\log^* n)$, if one considers Monte-Carlo algorithms with probability of success greater than one half [20]. Then the expected running time is also in $\Theta(\log^* n)$. This contrasts with the following result.

**Proposition 5.** *For 3-colouring on a ring, the expected complexity of an ordinary node is constant.*

*Proof.* The algorithm we consider, consists in repeating a simple procedure. At each round every node that has not yet an output, take a colour at random among the colours that are still available. That is, it takes a colour that as not yet been output by a neighbour. Note that this is always possible, as the nodes

have degree two, and choose among three colours. After the sampling, if there is no conflict, then the node outputs the colour. If there is a conflict, then the colour is forgotten, and the node continue to the next round. If the node outputs a colour, we say that it *succeeds*, otherwise it *fails*.

Given an arbitrary partial colouring obtained after some rounds, the probability that a fixed node succeeds is lower bounded by $\alpha = 5/12$. This number is obtained by case analysis. It corresponds to the case where, the current node has both neighbours without outputs, but both nodes at distance two with outputs, and these outputs are different. Let $\beta = 1 - \alpha$. Also, let $V_k$ be the number of nodes that have not yet output after round $k$, with $V_0 = n$. The following holds by linearity of the expectation.

$$\mathbb{E}(|V_k| \mid |V_{k-1}|) = \sum_{v \in V_{k-1}} \mathbb{P}(v \text{ does } \textit{not} \text{ stop at round } k) \leq \beta |V_{k-1}|$$

We can apply the previous inequality repeatedly, and get: $\mathbb{E}(V_k) \leq \beta^k n$. The number of nodes that stop at round $k$ is precisely $V_k - V_{k-1}$, then the sum of the running times is:

$$\sum_k k(V_k - V_{k-1}) \leq \sum_k kV_k.$$

The expected sum of the running time is then upper bounded by $\sum_k k\beta^k n$. Then the node-averaged expected sum is $\sum_k k\beta^k$. As $\beta < 1$, $\sum_k k\beta^k$ is a constant, thus the expected complexity of an ordinary node in a random ID assignment is constant. □

Note that having a constant complexity when looking at a more local measure, is not particular to this example. For example in [12], the author designs an algorithm for maximal independent set that terminates after $O(\log deg(v) + \log(1/\epsilon))$ rounds, with probability at least $1 - \epsilon$, where $deg(v)$ is the degree of node $v$.

## 6 Conclusion and Open Questions

This paper introduces the notions of node-averaged and ID-averaged complexities. We think these measures are meaningful when analysing algorithm that do not have the knowledge of the size of the network, or in contexts where partial solutions are useful. Also, very local complexities, as the one of Subsect. 5.1 and the one advocated in [12], are natural measures that one would like to understand better. Our results illustrate that these complexities can have interesting behaviours. The natural next step is to generalize these results, to more problems, and larger graph classes.

**Acknowledgements.** I would like to thank Juho Hirvonen, Tuomo Lempiäinen and Jukka Suomela for fruitful discussions, and Pierre Fraigniaud for both discussions, and help for the writing. I thank the reviewers for helpful comments, and Mohsen Ghaffari for pointing out that randomized node-averaged complexity could be considered.

## References


1. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. J. Algorithms **7**(4), 567–583 (1986)
2. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations, and Advanced Topics. Wiley, Hoboken (2004)
3. Barenboim, L., Elkin, M.: Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. Distrib. Comput. **22**(5–6), 363–379 (2010). https://doi.org/10.1007/s00446-009-0088-2
4. Brandt, S., Fischer, O., Hirvonen, J., Keller, B., Lempiäinen, T., Rybicki, J., Suomela, J., Uitto, J.: A lower bound for the distributed lovász local lemma. In: Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18–21, 2016, pp. 479–488 (2016). https://doi.org/10.1145/2897518.2897570
5. Brandt, S., Hirvonen, J., Korhonen, J.H., Lempiäinen, T., Östergård, P.R.J., Purcell, C., Rybicki, J., Suomela, J., Uznanski, P.: LCL problems on grids. CoRR, abs/1702.05456 (2017). arXiv:1702.05456
6. Chang, Y.J., Kopelowitz, T., Pettie, S.: An exponential separation between randomized and deterministic complexity in the local model. In: IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9–11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA, pp. 615–624 (2016). https://doi.org/10.1109/FOCS.2016.72
7. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Inf. Control **70**(1), 32–53 (1986). https://doi.org/10.1016/S0019-9958(86)80023-7
8. Feuilloley, L.: Brief announcement: average complexity for the LOCAL model. In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21–23, 2015, pp. 335–337 (2015). https://doi.org/10.1145/2767386.2767446
9. Feuilloley, L., Fraigniaud, P.: Survey of distributed decision. Bull. EATCS 119 (2016). EATCS: The Distributed Computing Column by Stefan Schmid
10. Fraigniaud, P., Korman, A., Peleg, D.: Towards a complexity theory for local distributed computing. J. ACM **60**(5), 35 (2013). https://doi.org/10.1145/2499228
11. Gamarnik, D., Sudan, M.: Limits of local algorithms over sparse random graphs. In: Proceedings of Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12–14, 2014, pp. 369–376 (2014). https://doi.org/10.1145/2554797.2554831
12. Ghaffari, M.: An improved distributed algorithm for maximal independent set. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10–12, 2016, pp. 270–277 (2016). https://doi.org/10.1137/1.9781611974331.ch20
13. Goldreich, O.: Introduction to testing graph properties. In: Goldreich, O. (ed.) Property Testing - Current Research and Surveys. LNCS, vol. 6390, pp. 105–141. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16367-8_7
14. Harris, D.G., Schneider, J., Su, H.H.: Distributed $(\Delta+1)$-coloring in sublogarithmic rounds. In: Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18–21, 2016, pp. 465–478 (2016). https://doi.org/10.1145/2897518.2897533
15. Korman, A., Sereni, J.-S., Viennot, L.: Toward more localized local algorithms: removing assumptions concerning global knowledge. Distrib. Comput. **26**(5–6), 289–308 (2013). https://doi.org/10.1007/s00446-012-0174-8

16. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
17. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**(4), 1036–1053 (1986). https://doi.org/10.1137/0215074
18. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, Burlington (1996)
19. Musto, T.: Knowledge of degree bounds in local algorithms. Master's thesis, University of Helsinki (2011)
20. Naor, M.: A lower bound on probabilistic algorithms for distributive ring coloring. SIAM J. Discrete Math. **4**(3), 409–412 (1991). https://doi.org/10.1137/0404036
21. Naor, M., Stockmeyer, L.J.: What can be computed locally? SIAM J. Comput. **24**(6), 1259–1277 (1995). https://doi.org/10.1137/S0097539793254571
22. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM, Philadelphia (2000)
23. Pettie, S., Ramachandran, V.: Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6–8, 2002, San Francisco, CA, USA., pp. 713–722 (2002). acm:545381.545477
24. Santoro, N.: Design and Analysis of Distributed Algorithms, vol. 56. Wiley, Hoboken (2006)
25. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences. A000788
26. Yao, A.C.C.: Probabilistic computations: toward a unified measure of complexity (extended abstract). In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, pp. 222–227 (1977). https://doi.org/10.1109/SFCS.1977.24

# How to Choose Friends Strategically

Lata Narayanan(✉) and Kangkang Wu

Department of Computer Science and Software Engineering,
Concordia University, Montreal, Canada
lata@encs.concordia.ca

**Abstract.** Alice wants to join a new social network, and influence its members to adopt a new product or idea. Each person $v$ in the network has a certain threshold $t(v)$ for *activation*, i.e. adoption of the product or idea. If $v$ has at least $t(v)$ activated neighbors, then $v$ will also become activated. If Alice wants to make $k$ new friends in the network, and thereby activate the most number of people, how should she choose these friends? We study the problem of choosing the $k$ people in the network to befriend, who will in turn activate the maximum number of people. This *Maximum Influence with Links* Problem has applications in viral marketing and the study of epidemics. We show that the solution can be quite different from the related and widely studied influence maximization problem where the objective is to choose a seed or target set with maximum influence. We prove that the Maximum Influence with Links problem is NP-complete even for bipartite graphs in which all nodes have threshold 1 or 2. In contrast, we give polynomial time algorithms that find optimal solutions for the problem for trees, paths, cycles, and cliques.

## 1 Introduction

The strategy of *viral marketing* for promoting new products or ideas is based on the observation that once a certain fraction of a social network adopts a product, we can expect a cascade of further adoptions [3,16,27]. Domingos and Richardson [12,33] were the first to raise the following important algorithmic problem in the context of social network analysis: If a company can turn a subset of customers in a given network into early adopters, and the goal is to trigger a large cascade of further adoptions, which set of customers should they target?

The social network can be modelled by a node-weighted graph $G = (V, E, t)$ with $V(G)$ representing individuals in the social network, $E(G)$ denoting the social connections, and $t$ an integer-valued *threshold function*. Starting with an initial *seed set or target set*, that is, a subset $S \subseteq V$ of nodes in the graph, that are *activated* by some external incentive, influence propagates deterministically in discrete time steps, and is said to *activate* nodes. For any unactivated node $v$, if the number of its activated neighbors at time step $i-1$ is at least $t(v)$,

Research supported by NSERC, Canada.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 283–302, 2017.
https://doi.org/10.1007/978-3-319-72050-0_17

then node $v$ will be activated in step $i$. A node once activated stays activated. Clearly the process terminates after at most $|V| - 1$ steps. We call the set of nodes that are activated when the process terminates as the *activated set*. The problem proposed by Domingo and Richardson [12,33] can now be formulated as follows: Given a social network $G = (V, E, t)$, and an integer $k$, find a subset $S \subseteq V$ of size $k$ so that the resulting activated set is as large as possible. In the context of viral marketing, the parameter $k$ corresponds to the budget, and $S$ is a target set that maximizes the size of the activated set. This influence maximization problem has been widely studied [2,5,14,17–19,22,28].

Recent work [10,20,25] points to some flaws in the above model for viral marketing. The authors of [10,20] observe that a limitation of the model is that there is no possibility of giving *partial* external incentives; indeed the initial seed set is activated *wholly* by external incentives, and the remaining nodes *only* by the internal network effect. The authors of [25] further critique the fact that the nodes in the seed set are assumed to be activated immediately by external incentives, *regardless of their own thresholds of activation*. They point out that this is unrealistic assumption, and suggest that it is likely that highly influential nodes have high thresholds, and cannot be activated by external incentives alone.

In this paper, we provide a new way to model a *viral marketing strategy with a fixed budget*, that addresses the flaws mentioned above. In particular, we model strategies in which each node in the target set is given some partial incentive, eg. a $10 coupon; for some people this may be enough for them to buy the product, for others, it reduces their resistance to buying it. We represent the initiator of the viral marketing strategy by a node *external to the social network*. We can now restate the influence maximization problem as follows: Suppose Alice, the external initiator, wants to join a new social network, and can make $k$ friends, how should she choose these friends if her goal is to influence as many people as possible? In other words, to which $k$ people already in the social network should Alice create links, so that she can activate the maximum number of people? If Alice creates a link to a node $v$, the threshold of $v$ is only effectively reduced by one, and so $v$ in turn is activated only if its threshold is one. We call our problem the *Maximum Influence with Links* problem (Max-Inf-Links).

Notice that the links added from the external node Alice correspond to the external incentive given to the endpoints of these links. The nodes that are the endpoints of these new links may not be immediately completely activated, but their thresholds are effectively reduced; this corresponds to their receiving partial incentives. Individuals with high thresholds cannot be activated only by external incentives, which better models reality. The Max-Inf-Links problem also has applications in epidemiology or the spread of diseases: if an infected person arrives from *outside* a community, the Max-Inf-Links problem corresponds to identifying the set of $k$ people such that if the infected external person has contact with this set, the highest number of people in the community could potentially be infected. The problem can obviously be generalized to a set of external influencers wishing to infect or gain control of a network.

Observe that the solution to the Max-Inf-Links problem can be quite different from the solution to choosing an optimal seed set for a given network. For example, consider a clique $K_n$ in which node 1 has threshold $n-1$ and the remaining nodes have threshold 1. If the budget $k$ is 1, the optimal target set is clearly the node $x$, as choosing it activates the entire network. However, if we choose to *link* to the node $x$, it does not get activated, since its threshold is $n-1$, and therefore no node in the network gets activated. The optimal solution to the Max-Inf-Links problem is in fact to choose any of the other nodes in $K_n$; this would activate the entire network. This would however be a sub-optimal seed set.

### 1.1 Our Results

It was shown recently that it is NP-hard to find the minimum number of links required for an external influencer to activate the entire network [25]. Since this is a special case of our Max-Inf-Links problem, it follows that our problem is NP-hard in general. We prove here that the Max-Inf-Links problem is NP-hard, even if all nodes have threshold 1 or 2, even in bipartite graphs, which have many applications in social networks. Note that the NP-hardness proof in [25] does not apply to bipartite graphs. In light of the hardness result, we study the complexity of the problem for social networks that can be represented as trees, cycles, and cliques. We show that optimal solutions can be found for the Max-Inf-Links problem with $k$ links in time $\Theta(kn)$ for paths, $\Theta(kn^2)$ for cycles, $\Theta(k^2n^2)$ for trees, and $\Theta(n)$ for cliques. Note that $k$ is always upper bounded by $n$. Our algorithms for paths, trees, and cycles rely on non-trivial dynamic programming formulations. Note that the algorithm for trees can be used for paths, but we are able to get a much faster implementation for paths by making careful use of the solution properties.

### 1.2 Related Work

The problem of identifying the most influential nodes in a social network has received a tremendous amount of attention [2,5,14,17–19,22,28]. The algorithmic question of choosing a seed set of size $k$ that activates the most number of nodes in the context of viral marketing was first posed by Domingos and Richardson [12]. Many natural heuristics were proposed for the problem, such as choosing the nodes of highest degree, or those central in terms of distance [23,24]. Kempe et al. [23] started the study of this problem as a discrete optimization problem, and studied it in both the probabilistic independent cascade model and the threshold model of the influence diffusion process. They showed the NP-hardness of the problem in both models, and showed that a natural greedy strategy has a $(1 - 1/e - \epsilon)$-approximation guarantee in both models; these results were generalized to a more general cascade model in [24]. Mossel and Roch [30] further generalized the results of [23,24] by positively resolving their conjecture that whenever the local threshold functions are local and submodular, the resulting influence function is also a submodular function. Borgs *et al.* [2] recently obtained a significant improvement by giving an algorithm

that obtains an approximation factor of $1 - 1/e - \epsilon$ for any $\epsilon > 0$, in time $O((m+n)k\log(n)/\epsilon^2)$. A large body of work studies algorithms that have performance guarantees and at the same time scale in practice to real-life and very large-scale social networks [31,35,36]. Influence diffusion under time window or deadline constraints has also been studied [11,15,26,29].

In the Target Set Selection problem [1,4,32], the size of the target set or budget is not specified in advance, but the goal is to activate the entire network or a fixed fraction of nodes. Partial incentives were studied in the context of the target set selection problem in [8,10,20,21,25]. The closest formulation to our paper is the *Minimum Links* problem introduced in [25], where it is required to find the minimum number of links an external influencer needs to form to nodes in the network so that the entire network is eventually activated. It was shown in [25] that the Minimum Links problem is NP-hard, and in fact, it is even hard to approximate with ratio $\epsilon$ for some constant $\epsilon$. The authors gave linear time greedy algorithms for trees, cycles, and cliques.

Demaine *et al.* [10] were the first to introduce the study of the maximization of influence with partial incentives and a fixed budget [10]. However, they consider thresholds chosen uniformly at random, while we study arbitrary thresholds. Additionally, they allow arbitrary fractional influence to be applied externally on any node, while in our model, every node that receives a link has its threshold reduced by the same amount. Eftekhar et al. studied a model where nodes become seed nodes with a fixed probability [13]. Recently, this idea was extended by considering the idea of offering discounts to nodes, which would cause some nodes to be activated with a probability proportional to the amount of the discount [34,37].

## 2 Notation and Preliminaries

Given a social network represented by an undirected graph $G = (V, E, t)$, we introduce a set of external nodes $U$ that are assumed to be already activated. We assume that all edges have unit weight; this is generally called the *uniform weight assumption*, and has previously been considered in many papers [4,6,7,15,25]. A *link set* for $(G, U)$ is a set $S$ of links between nodes in $U$ and nodes in $V$, i.e. $S \subseteq \{(u, v) \mid u \in U; v \in V\}$. For a link set $S$, we define $E(S) = \{v \in V \mid \exists (u, v) \in S\}$, that is, $E(S)$ is the set of $V$-endpoints of links in $S$. For a node $v \in V(G)$, define $r(v)$ to be the number of links in $S$ for which $v$ is an endpoint. Since the set of external nodes $U$ is already activated, observe that adding the link set $S$ to $G$ is equivalent to reducing the threshold of the node $v$ by $r(v)$. In the viral marketing scenario, the link set $S$ represents giving $v$ a partial incentive of $r(v)$.

Given a link set $S$ for a graph $G$, we define $I(G, S)$ to be the set of nodes in $G$ that are eventually activated as a result of adding the link set $S$, that is, by reducing the threshold of each node $v \in E(S)$ by $min\{r(v), t(v)\}$, and then running the influence diffusion process. See Fig. 1 for an illustration. Observe that this is the same as the set of nodes activated by using $U$ as the target set

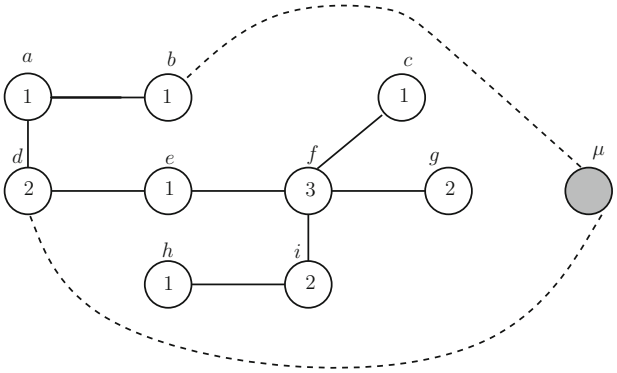



**Fig. 1.** Node $\mu$ is the external influencer and is assumed to be activated. Links in the link set are shown with dashed edges. The given link set is an optimal link set of size 2 that activates 4 nodes: $a, b, d, e$.

in the graph $G'$, the graph obtained from $G$ by adding the set $U$ to the vertex set and the set $S$ to the set of edges.

**Definition 1. Maximum Influence with Links problem** (Max-Inf-Links)**:** *Given a social network $G = (V, E, t)$, and a set of external nodes $U$, and an integer $k$, find a link set $S$ of size $k$ that maximizes $I(G, S)$ among all $k$-sized link sets. We denote by $MI(G, k)$ the maximum number of people that can be influenced in $G$ by an optimal link set of size $k$, that is, if $S$ is an optimal solution to the* Max-Inf-Links*$(G, k)$ problem, then $MI(G, k) = |I(G, S)|$.*

In our algorithms, we consider the case of a *single influencer*, that is, $U = \{\mu\}$. In this case, a link given to a vertex $v$ reduces its threshold by 1. Since $\mu$ must be an endpoint of each edge in the link set $S$, each such edge can be uniquely specified by a vertex in $V$. We therefore generally omit mention of $\mu$ in the rest of the paper, and the link set $S$ is referred to by its $V$-endpoints. For each such node $v \in E(S)$, we say we *give* $v$ a link, or that $v$ *receives* a link.

## 3 NP-Hardness of Max-Inf-Links

In this section, we will prove that the decision version of the Max-Inf-Links problem is NP-hard even for bipartite graphs in which thresholds of all nodes are either one or two. It can be posed as follows: Given a social network $G = (V, E, t)$, a set of external influencers $A$, and integers $k$ and $p$, is there a link set $S$ of size $k$ such that $|I(G, S)| = p$? Note that in [25], the Min-Links problem (finding the smallest set of links which can activate the entire network) was shown to be NP-hard, even for graphs of degree 3 and threshold at most 2. This immediately implies the NP-hardness of the Max-Inf-Links problem as well. However, the reduction in [25] yielded a graph that was not bipartite. Here, we extend the result for bipartite graphs.

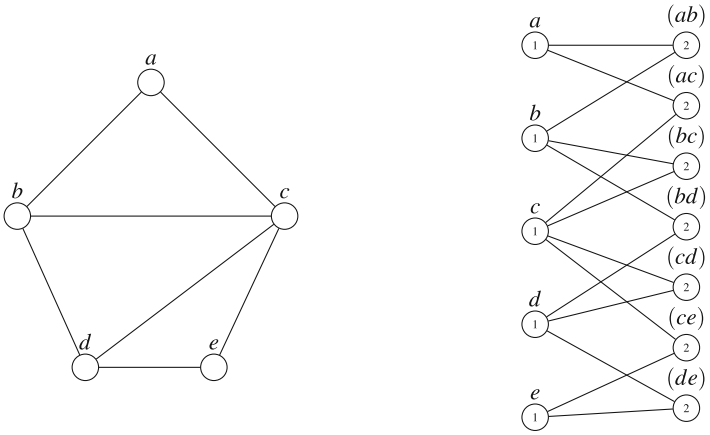



**Fig. 2.** Reduction from $G$ (top) to $G'$ (bottom); thresholds in $G'$ are indicated inside the circles representing the nodes.

**Theorem 1.** *The decision version of the* Max-Inf-Links *problem is NP-hard, even for a single external influencer, and bipartite graphs with all nodes having threshold 1 or 2.*

*Proof.* We give a reduction from the *Max-Clique problem*: Given a graph $G = (V, E)$ and an integer $k$, does $G$ contain a clique of size at least $k$?

Given an instance of the Max-Clique problem $(G, k)$, we construct a bipartite graph $G' = (V_1 \cup V_2,\ E',\ t)$ as follows. For every node $v \in V$, we create a corresponding node $v$ of threshold 1 in $V_1$. For every edge $\{u, v\} \in E$, we create a corresponding node $(uv)$ of threshold 2 in $V_2$. Next, for every edge $\{u, v\} \in E$, we create the edges $(u, (uv))$ and $(v, (uv))$. Clearly, the transformation can be done in $O(V + E)$ time. We show that $G$ has a clique of size $k$ if and only if $G'$ has a link set of size $k$ that can activate at least $C_k^2 + k$ nodes.

We show that $G$ has a clique of size $k$ if and only if $G'$ has a link set of size $k$ that can activate at least $C_k^2 + k$ nodes. Figure 2 illustrates the reduction.

To do this, we first show that it suffices to consider link sets that contain only nodes in $V_1$.

*Claim.* For any link set $T \subseteq V_1 \cup V_2$, there exists a set $S \subseteq V_1$ such that $|S| \leq |T|$ and $|I(G', S)| \geq |I(G', T)|$.

*Proof.* Consider a node $v \in V_2$ that receives a link in $T$, and is connected to $v_1$ and $v_2 \in V_1$. We argue that we can either remove the link assigned to $v$, or assign that link to some other node in $V_1$ while not decreasing the size of the activated set. The following four cases about the time of activation of $v$ are exhaustive.

Case 1: $v$ is not activated by $T$: We can simply remove the link assigned to $v$.
Case 2: $v$ is activated before $v_1$ and $v_2$: Since $v$ has threshold 2 while $v_1$ and $v_2$ are both of threshold 1, this is impossible with a single influencer, as the link $v$ receives can only reduce its threshold by 1.

Case 3: $v$ is activated after $v_1$ (or $v_2$) is activated and before $v_2$ (rest. $v_1$) is activated: We can move the link assigned for $v$ to $v_2$ (resp. $v_1$); the same set of nodes will be activated eventually.

Case 4: $v$ is activated after $v_1$ and $v_2$ are activated: The link given to $v$ is unnecessary and can be removed.

*Claim.* $G'$ has a link set $S \subseteq V_1$ with $|S| \leq k$ and $|I(G', S)| \geq C_k^2 + k$ if and only if $G$ has a clique of size $k$.

*Proof.* Suppose first that $G$ has a clique $V'$ of size $k$. We claim that $V' \subseteq V_1$ is a link set such that $|I(G', V')| = C_k^2 + k$. Clearly, all nodes in $V'$ are activated at round 1. Then since $V'$ is a clique in $G$, it is easy to see that all nodes in $V_2$ corresponding to the $C_k^2$ edges between nodes in $V'$ will be activated in round 2. This proves that $|I(G', V')| \geq C_k^2 + k$ .

Suppose next that there is a link set $S \subseteq V_1$ such that $|I(G', V')| = C_k^2 + k$, then we claim that the corresponding set for $S$ in $G$ forms a clique of size $k$.

We claim that given $S \subseteq V_1$ as a link set, it is impossible to activate any new node $\in V_1$ except nodes in $S$ and for any node $(uv) \in V_2$ that gets activated, it must be that $u \in S$ and $v \in S$.

Suppose there exists a node $d \in V_1 - S$ such that $d$ is eventually activated. Since $d$ has threshold 1 and $d$ doesn't receive a link, in order for $d$ to be activated, one of $d'$s neighbors must be activated first. Let us say $d$ is connected to $(dx)$ and $(dx)$ is activated before $d$. Node $(dx)$ is of threshold 2, thus in order for $(dx)$ to be activated, both $d$ and $x$ must be activated before $(dx)$, a contradiction.

For the second part, suppose there exists a node $(uv) \in V_2$ such that $u \notin S$ and $(uv)$ gets activated eventually. But from the previous analysis, we know that $u$ can never be activated. Therefore $(uv)$ cannot be activated either.

We have shown that with a set $S \subseteq V_1$ of size $k$, only nodes in $S$ can be activated for the nodes in $V_1$; only those nodes which are connected to two nodes in $S$ can be activated for nodes in $V_2$. If $|I(G', S)| \geq C_k^2 + k$, then for every pair of nodes in $S$, they must be connected in $G$, thus $S$ must be a clique of size $k$ in $G$.

This completes the proof of the reduction. □

## 4 Optimal Algorithm for Trees

In contrast to the result of the previous section, in this section, we show that the Max-Inf-Links problem can be solved in polynomial time in trees. Let $T = (V, E, t)$ be a tree with $n$ nodes, $V = \{1, 2, ..., n\}$ and $t : t(v) \rightarrow \mathcal{Z}^+$. Fix an arbitrary root of the tree and order the children of every node in an arbitrary fashion. We define $T_v$ and $d_v$ to be the sub-tree rooted at node $v$, and the number of children of node $v$ respectively. We define $T_v^{-1}$ to be the same sub-tree as $T_v$ except that the threshold of the root $v$ is reduced by 1. Note that while in the input, all thresholds are $\geq 1$, in a tree $T_v^{-1}$, the threshold of $v$ may be reduced to 0. We also define $v_i$ to be $i^{th}$ child of node $v$ and $T_{v_i}$ to be the sub-tree rooted at $v_i$.

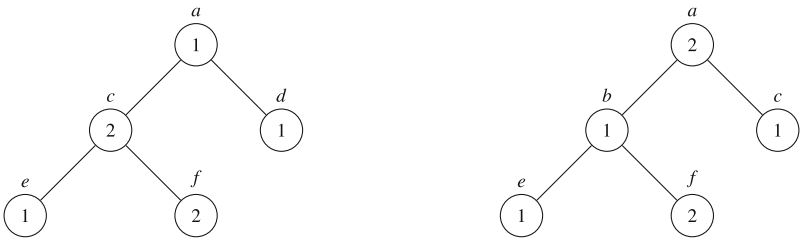



**Fig. 3.** For the tree on the left, $A(T, 1) = 2$, and $B(T, 1) = 1$ while for the tree on the right, $A(T, 1) = -\infty$, and $B(T, 1) = 1$

Given a tree $T_v$, the optimal solution to the Max-Inf-Links problem for $T_v$ may or may not activate the root. For example, in Fig. 3, for the tree on the left, it can be verified that every optimal link set of size 1 activates the root (one such link set gives a link to the root node $a$ and activates $a$ and $d$), but for the tree on the right, there is no link set of size 1 that activates the node $a$ (an optimal solution gives a link to node $b$, and activates both $b$ and $c$) . Let $MIA(T_v, k)$ be the problem of finding an link set of at most $k$ links to nodes in the sub-tree $T_v$ that maximizes the number of influenced nodes *while ensuring that root $v$ is activated*. Similarly, let $MIB(T_v, k)$ be the problem of finding an link set of at most $k$ links to nodes in the sub-tree $T_v$ that maximizes the number of influenced nodes *while ensuring that root $v$ is not activated*. Let $A(T_v, k)$ and $B(T_v, k)$ be the maximum number of nodes that can be influenced by an optimal solution to $MIA(T_v, k)$ and $MIB(T_v, k)$ respectively. Clearly, given a tree $T = (V, E, t)$ rooted at node $r$, an optimal link set $S$ for problem Max-Inf-Links$(T_n, k)$ either activates or does not activate the root $r$. Therefore:

$$MI(T_n, k) = max\{A(T_r, k), B(T_r, k)\}$$

We start with link sets that do activate the root. We first prove a critical lemma that shows that any link set $S$ that activates the root can be converted to an equivalent link set that gives a link to the root.

**Lemma 1.** *Suppose $S$ is a link set of size $k$ for a tree $T_v$ in which $v = root(T)$ is activated, and $(\mu, v) \notin S$. Then there exists a link set $S'$ with $(\mu, v) \in S'$ such that $I(T, S') = I(T, S)$ and $|S'| = k$.*

*Proof.* We prove the lemma by induction on the height of the tree $T$. Clearly, the lemma is true for trees of height 0; the only way to activate the root in such a tree is to give a link to the root $v$.

Now consider a tree $T$ of height $h$ and a link set $S$ for $T$ that activates $v$ without giving $v$ a link. Clearly, there must exist a child $c$ of $v$ which is activated before $v$, and contributes to the activation of $v$. By the inductive hypothesis, we can assume that $c$ is given a link in $S$. Consider $S' = S - \{(\mu, c)\} \cup \{(\mu, v)\}$. Any node in $T_v - T_c$ that was activated by $S$ before $v$, is also activated in $S'$ before $v$. Therefore $v$ will be activated by $S'$, and will subsequently activate $c$, and any

nodes in $T_c$ that were activated by $c$. All other children of $v$ that were activated after $v$ in $S$ will also be activated after $v$ by $S'$. Therefore, $I(T,S) = I(T,S')$. □

Lemma 1 tells us that an optimal link set $S$ of size $k$ that activates the root $v$ of a tree $T_v$ can be assumed to give a link to the root $v$. Then the remaining $k-1$ links must be given to other nodes in $T_v$, but since $v$ is activated by $S$, it must be that at least $t(v)-1$ children of $v$ are activated *before* $v$ and contribute to the activation of $v$. Observe that once $v$ is activated, the thresholds of its remaining children are effectively reduced by 1. This, together with the links given by the link set $S$ to nodes in the subtrees of the remaining children, may activate some of these children and subsequently other nodes in their subtrees. Let $F_{v,d}$ be the forest of subtrees rooted at the first $d$ children of $v$. We see that there exists some partition $(C,D)$ of the roots of trees in $F_{v,d_v}$ such that $|C| \geq t(v)-1$ and $S$ activates the nodes in $C$ before $v$, and subsequently $v$ gets activated, which in turn reduces the thresholds of the nodes in $D$ by one, perhaps contributing to their activation.

We formalize and generalize the above to find appropriate link sets for the forest $F_{v,d_v}$. Let $F_{v,d}$ be the forest consisting of the subtrees rooted at the first $d$ children of $v$ and let $(C,D)$ be a partition of the roots of the $d$ trees. Let $F(C,D)$ be a forest derived from $F_{v,d}$ by reducing the threshold of all nodes in $D$. We call a link set for $F(C,D)$ a $C$-activating link set if it activates all nodes in $C$, and maximizes the total number of activated nodes in $F(C,D)$. Given a forest $F$, and an integer $i$, we call a link set for $F$ an $i$-first link set if it is a $C$-activating link set for some partition $(C,D)$ of the roots of trees in $F$ with $|C| \geq i$. We now define $MIA(F_{v,d}, i, k)$ to be the problem of finding an $i$-first link set of size $k$ that maximizes the total number of activated nodes in $F$.

Next consider a link set that does not activate $v$. In this case, at most $t(v)-1$ of $v$'s children can be activated, as otherwise $v$ would also be activated. To find such a link set, we define $MIB(F_{v,d}, i, k)$ to be the problem of finding a link set of size at most $k$ that activates the most nodes in $F_{v,d}$ while activating exactly $i$ roots of trees in $F_{v,d}$. Let $A(F_{v,d}, i, k)$ and $B(F_{v,d}, i, k)$ be the number of influenced nodes by an optimal solution to $MIA(F_{v,d}, i, k)$. and $MIB(F_{v,d}, i, k)$ respectively.

We now give recursive formulations for $A(T_v, k)$, $B(T_v, k)$, $A(F_{v,d}, i, k)$, and $B(F_{v,d}, i, k)$; as we have already observed, they are inter-dependent. We start with a recursive formulation for $A(v, k)$.

**Lemma 2**

$$A(T_v,k) = \begin{cases} 1+A(F_{v,d_v},0,k) & \text{if } t(v)=0 \\ 1+A(F_{v,d_v},t(v)-1,k-1) & \text{if } 1 \leq t(v) \leq k \\ -\infty & \text{if } k < t(v) \end{cases}$$

*Proof* First suppose $k \geq t(v) = 0$. In this case, there is no need to give $v$ a link, and none of $v$'s children need to be activated before it. Therefore, $A(T_v, k) = 1 + A(F_{v,d_v}, 0, k)$ as claimed.

Next suppose $k \geq t(v) = 1$. Then by Lemma 1, there exists an optimal link set $S$ in which root $v$ receives a link, thereby activating $v$. It is then straightforward to see that $S - \{v\}$ is also an optimal solution to problem $MIA(F_{v,d_v}, 0, k-1)$. It follows that $A(T_v, k) = 1 + A(F_{v,d_v}, 0, k-1)$ in this case.

Next, suppose $1 < t(v) \leq k$. Then giving root $v$ a link does not suffice to activate $v$; in fact $t(v)-1$ of $v$'s children have to be activated before $v$, otherwise $v$ cannot be activated. By Lemma 1, however, there exists an optimal link set $S$ which gives root $v$ a link. We claim that $S-\{v\}$ is an optimal solution to problem $MIA(F_{v,d_v}, t(v)-1, k-1)$. Suppose not, let $|S| = k$ and let $S'$ be an optimal solution to $MIA(F_{v,d_v}, t(v)-1, k-1)$ which can activate more nodes than $S - \{v\}$. With $S'$, $t(v)-1$ children of $v$ have been activated. These activations, together with the link to $v$, are enough to activate node $v$, and would reduce the threshold of the remaining children of $v$ by 1. Therefore $S' \cup \{v\}$ activates $v$ as well as all nodes activated by $S'$ in $F_{v,d(v)}$. Then $S' \cup \{v\}$ will be a link set of $k$ links which can activate more nodes in $T_v$ than $S$, contradicting the optimality of $S$. Therefore: $A(T_v, k) = 1 + A(F_{v,d_v}, t(v)-1, k-1)$.

Finally suppose $t(v) > k \geq 0$. Then we claim it is impossible to activate the root. If it were possible to activate the root using $k$ links, then by Lemma 1, there is an optimal link set that activates the root and gives a link to $v$. Then the remaining $k-1$ links must be given to nodes in $F_{v,d_v}$. However using $k-1 < t(v)-1$ links, we can activate at most $k-1$ children of $v$, and together with the link given to $v$, the reduction in threshold of $v$ is $< t(v)$, a contradiction. Thus in this case $A(T_v, i) = -\infty$ as claimed. This completes the proof of the lemma. □

**Lemma 3.** $A(F_{v,d}, i, k)$

$$= \begin{cases} -\infty & \text{if } i > d \\ 0 & \text{if } i = d = 0 \\ \max\limits_{0 \leq p \leq k} \{A(F_{v,d-1}, i-1, p) + A(T_{v_d}, k-p)\} & \text{if } i = d > 0 \\ \max \left\{ \begin{array}{l} \max\limits_{0 \leq p \leq k} A(F_{v,d-1}, i, p) + max \left\{ \begin{array}{l} A(T_{v_d}^{-1}, k-p) \\ B(T_{v_d}^{-1}, k-p) \end{array} \right. \\ \max\limits_{0 \leq p \leq k} \{A(F_{v,d-1}, i-1, p) + A(T_{v_d}, k-p)\} \end{array} \right. & \text{if } 0 < i < d \\ \max\limits_{0 \leq p \leq k} A(F_{v,d-1}, i, p) + max \left\{ \begin{array}{l} A(T_{v_d}^{-1}, k-p) \\ B(T_{v_d}^{-1}, k-p) \end{array} \right. & \text{if } 0 = i < d \end{cases}$$

*Proof.* If $i > d$, clearly it is impossible to activate at least $i$ of the first $d$ children of $v$, therefore $A(F_{v,d}, i, k) = -\infty$. If $i = d = 0$, the forest $F_{v,d}$ is empty, and therefore, no nodes can be influenced by any link set of any size.

If $i = d$, this means that all children $v_j$ with $1 \leq j \leq d$ have to be activated before $v$. The optimal solution $S$ will assign $p$ links to $F_{v,d-1}$ and $k-p$ links to $T_{v_d}$, for some $p$. It follows that $A(F_{v,d}, i, k) = \max\limits_{0 \leq p \leq k} \{A(F_{v,d-1}, i-1, p) + A(T_{v_d}, k-p)\}$ If $i < d$, and $v_d$ is not activated before $v$ by $S$, observe that $v_d$ may or may not

be activated after $v$. Besides, due to the fact that its parent has already been activated, the threshold of $v_d$ is effectively reduced by one. An optimal link set $S$ will assign $p$ links to $F_{v,d-1}$ and $k-p$ links to $T_{v_d}$, therefore we try all possibilities of $p$ to find the best distribution. Therefore:

$$A(F_{v,d},i,k) = \max_{0\le p\le k}\{A(F_{v,d-1},i,p) + max\begin{Bmatrix} A(T_{v_d}^{-1},k-p) \\ B(T_{v_d}^{-1},k-p)\end{Bmatrix}\}$$

Finally, if $i < d$ and $v_d$ is activated before $v$ by $S$, at least $i-1$ children of $v$ in $F_{v,d-1}$ are required to be activated before $v$. Since $v_d$ contributes to the activation of $v$, the threshold of $v_d$ remains unchanged.

$$A(F_{v,d},i,k) = \max_{0\le p\le k}\{A(F_{v,d-1},i-1,p) + A(T_{v_d},k-p)\}$$

When $i = 0$, we only need to consider the first of the two situations above. □

Next we consider link sets that do not activate the vertex $v$ (the proof of Lemma 4 is omitted).

**Lemma 4**

$$B(T_v,k) = \begin{cases} \max\limits_{0\le i<min(t(v),d_v+1)} B(F_{v,d_v},i,k) & \text{if } t(v) > 0 \\ -\infty & \text{if } t(v) = 0 \end{cases}$$

**Lemma 5.** $B(F_{v,d},i,k)$

$$= \begin{cases} \max\begin{cases} \max\limits_{0\le p\le k}\{B(F_{v,d-1},i,p) + B(T_{v_d},k-p)\} \\ \max\limits_{0\le p\le k}\{B(F_{v,d-1},i-1,p) + A(T_{v_d},k-p)\}\end{cases} & \text{if } 0 < i,d \\ \max\limits_{0\le p\le k}\{B(F_{v,d-1},i,p) + B(T_{v_d},k-p)\} & \text{if } 0 = i < d \\ 0 & \text{if } d = 0 \end{cases}$$

*Proof.* Suppose $i,d > 0$. Then in an optimal solution $S$ to $MIB(F_{v,d},i,k)$, either $v_d$ is activated or it is not. If $v_d$ is activated, then some $p$ links are given to nodes in $F_{v,d-1}$ and exactly $i-1$ nodes are activated by $S$ in $F_{v,d-1}$, while the remaining $k-p$ links must constitute an optimal solution to $MIA(T_{v_d},k-p)$. That is, $B(F_{v,d},i,k) = B(F_{v,d-1},i-1,p)+A(T_{v_d},k-p)$. Alternatively, if $v_d$ is not activated by $S$, then some $p$ links are given to nodes in $F_{v,d-1}$ and exactly $i$ nodes are activated by $S$ in $F_{v,d-1}$, while the remaining $k-p$ links must constitute an optimal solution to $MIB(T_{v_d},k-p)$. That is, $B(F_{v,d},i,k) = B(F_{v,d-1},i,p) + B(T_{v_d},k-p)$. If $i = 0$, this means no roots of trees in $F_{v,d}$ are to be activated, which means neither $v_d$ is activated, nor any roots in $F_{v,d-1}$ . Finally, if the forest is empty ($d = 0$), then clearly the maximum number of nodes that can be influenced is 0, regardless of $i$ and $k$. □

**Theorem 2.** *The Maximum Influence problem for a tree $T_n = (V, E, t)$ using $k$ links, can be solved in time $O(n^2k^2)$.*

*Proof.* The recursive definitions given in Lemmas 2 to 5 can be computed using dynamic programming to obtain an optimal solution to the Max-Inf-Links problem. In the worst case, for any node $w$ with $d$ children, we need to compute $A(F_{v,p}, i, j)$ and $B(F_{v,p}, i, j)$ for $1 \leq p \leq d$, $0 \leq i \leq d$, and $0 \leq j \leq k$. These are $d^2k$ values, each of which can take $\Theta(k)$ time to compute. The values $A(T_v, k)$ and $B(T_v, k)$ can be computed in constant time, and there are $k$ such values. Using the result of [9], the sum of squares of degrees of a node in a graph is upper bounded by $e(2e/(n-1)+n-2)$, which for a tree is $\Theta(n^2)$. Summing up the time over all vertices in the graph, we obtain a total time of $O(n^2k^2)$. □

## 5 Faster Algorithm for Paths

The algorithm for trees in the previous section obviously also applies to paths. In this section, we give a $\Theta(kn)$ algorithm for the Max-Inf-Links problem in a path, by exploiting its simpler structure. Let $P_n = (V, E, t)$ be a path with $n$ nodes, $V = \{1, 2, ..., n\}$, $E = \{(i, (i+1)) \mid 1 \leq i \leq n-1\}$, and $t : t(v) \rightarrow \mathcal{Z}^+$ (the threshold function). For $1 \leq i \leq j \leq n$, we define $P_{i,j}$ to be the sub-path of $P_n$ consisting of all nodes in $\{i, \ldots, j\}$.

An optimal solution to Max-Inf-Links$(P_{i,j}, k)$ may or may not activate the node $i$. Accordingly, let $MIA(P_{i,j}, k)$ be the problem of finding a link set of size at most $k$ for the sub-path $P_{i,j}$ that maximizes the number of influenced nodes *while ensuring that node $i$ is activated*. Similarly, let $MIB(P_{i,j}, k)$ be the problem of finding a link set of size at most $k$ for the sub-path $P_{i,j}$ that maximizes the number of influenced nodes *while ensuring that node $i$ is not activated*. Let $A(i, j, k)$ and $B(i, j, k)$ be the number of nodes that can be influenced by optimal solutions to $MIA(P_{i,j}, k)$ and $MIB(P_{i,j}, k)$ respectively.

The key idea of our algorithm is to break the path into subpaths containing only nodes of threshold 1 and 2, separated by nodes of threshold 3 or greater. We give recursive definitions for $A(i, j, k)$ and $B(i, j, k)$ when $P_{i,j}$ has only nodes of threshold 1 or 2. As we will see, these definitions are inter-dependent. We need the following definition.

**Definition 2.** *Given a path $P_n$, fix $i$ such that $1 \leq i \leq n$. We define $next(i) = min\{j \mid i < j \leq n+1$ and either $t(j) = 1$ or $j = n+1\}$.*

We see that $next(i)$ is the first node after $i$ to have threshold 1, unless $i$ is the rightmost node in the path with threshold 1, in which case $next(i) = n+1$. Clearly $A(n+1, n, k) = B(n+1, n, k) = 0$ for all $k$. We now consider the case when $1 \leq i \leq j \leq n$. We start with the case when $t(i) = 2$.

**Lemma 6.** *Given a sub-path $P_{i,j}$ in which all nodes have threshold 1 or 2, and $t(i) = 2$:*

$$A(i,j,k) = \begin{cases} 0 & \text{if } next(i) > j \\ 0 & \text{if } A(i+1,j,k-1) = 0 \\ 1 + A(i+1,j,k-1) & \text{if } A(i+1,j,k-1) > 0 \end{cases}$$

$$B(i,j,k) = \begin{cases} 0 & \text{if } next(i) > j \\ max \begin{cases} A(i+1,j,k) \\ B(i+1,j,k) \end{cases} & \text{if } next(i) \leq j \end{cases}$$

*Proof.* If $next(i) > j$ then there is no way to activate any node in $P_{i,j}$, therefore $A(i,j,k) = B(i,j,k) = 0$ for every $k$. If instead $next(i) \leq j$, then it is possible to activate at least one node in $P_{i,j}$. Observe that in any feasible solution for $MIA(P_{i,j},k)$, not only does node $i$ need to receive a link, but its neighbor node $i+1$ needs to be activated as well, therefore $A(i,j,k) = 0$ if $A(i+1,j,k-1) = 0$; otherwise $A(i,j,k) = 1+A(i+1,j,k-1)$. Finally, note that any feasible solution for $MIB(P_{i,j},k)$ is a solution in which $i$ does not receive a link, and the next node may or may not be activated, or it does receive a link, and the next node is not activated. That is $B(i,j,k) = max\{A(i+1,j,k), B(i+1,j,k), B(i+1,j,k-1)\} = max\{A(i+1,j,k), B(i+1,j,k)\}$. □

Next we consider the case when node $i$ has threshold 1. In this case, the optimal substructure of the problem is not so straightforward to prove. The difficulty arises because an optimal solution $S$ to $MIA(P_{i,j},k)$ may or may not activate node $i+1$. In the case when it does not activate node $i+1$, we would like to claim that $S$ consists of a link to $i$ and and an optimal solution to $MIB(P_{i+1,j},k-1)$. However, we do not know whether or not $i+2$ was activated in a solution to $MIB(P_{i,j},k)$. So when we combine such a solution with a link to node $i$, we may or may not activate node $i+1$.The following technical lemma uncovers the structure of optimal solutions.

**Lemma 7.** *Let $P_{i,j}$ be a path with $t(i) = 1$ and $next(i) \leq j$. If in every optimal solution for $MIA(P_{i,j},k)$, node $i$ receives a link, then there exists an optimal solution $S$ for $MIA(P_{i,j},k)$ in which neither $i+1$ nor $i+2$ receives a link.*

*Proof.* Suppose in every optimal solution for $MIA(P_{i,j},k)$, node $i$ receives a link. Let $S$ be an optimal solution for $MIA(P_{i,j},k)$ which uses the fewest links possible.

First we show that $next(i) > i+2$. By assumption, node $i$ receives a link in $S$. Observe that if $next(i) = i+1$, then $i+1$ cannot have a link since that would contradict the minimality of $S$. So we can simply move the link from node $i$ to $i+1$ and activate the same set of nodes, but this contradicts the assumption that in every optimal solution, node $i$ must receive a link. Next suppose $next(i) = i+2$. Then it is not possible that both $i+1$ and $i+2$ receive links, as this would contradict the minimality of $S$. If exactly one of $i+1$ and $i+2$ have a link, we

can move the link to $i$ to the node among $i+1$ and $i+2$ that does not have a link, thus creating a solution which activates exactly the same set of nodes, a contradiction. If neither $i+1$ nor $i+2$ has a link, and neither is activated, then the lemma is proved. Finally, if neither has a link, but one of them is activated, it must be that $i+2$ is activated by node $i+3$. In this case, we can move the link from node $i$ to node $i+1$, getting a solution that activates the same set of nodes, a contradiction to the assumption that every optimal solution must give $i$ a link. We conclude that $next(i) > i+2$, that is, there are at least two nodes in between $i$ and $next(i)$.

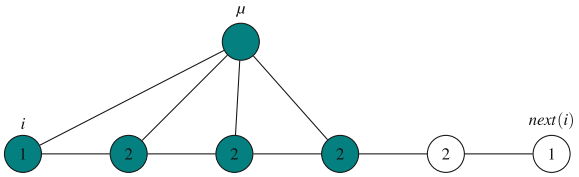



**Fig. 4.** The case when $next(i)$ is not activated. At least one node in $\{i+1, \ldots, next(i)-1\}$ does not receive a link.

We now show that we can always change $S$ to a solution $S'$ that does not activate $i+1$ or $i+2$ but activates the same number of nodes as $S$ overall. Let $S_1 = S \cap \{i+1, \ldots, next(i)-1\}$ and $S_2 = S \cap \{next(i), \ldots, j\}$. First suppose $next(i)$ is not activated by $S$. Then $next(i)$ did not get a link, and there must be at least one other node in $\{i+1, \ldots, next(i)-1\}$ that did not receive a link, as if all such nodes received a link, $next(i)$ would be activated (see Fig. 4). Therefore $|S_1| \leq next(i) - i - 2$. Consider now the link set $S'$ made by *shifting* all the links in $S_1$ from $i+1$ onwards to a consecutive sequence of nodes ending with $next(i)$, that is, $S' = \{i\} \cup \{next(i) - |S_1| + 1, \ldots next(i)\} \cup S_2$. Then $S'$ is the same size as $S$, it activates exactly the same number of nodes as $S$, and is therefore also optimal. Observe that $next(i) - |S_1| + 1 > i + 2$, since $|S_1| \leq next(i) - i - 2$. Thus $S'$ is an optimal solution to $MIA(P_{i,j}$ that does not give links to $i+1$ and $i+2$ as needed.

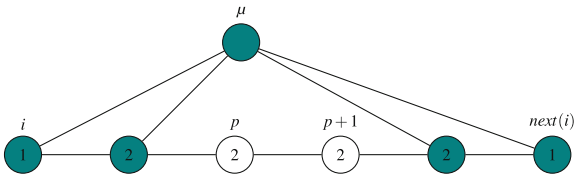



**Fig. 5.** The case when $next(i)$ is activated either by a link or by $next(i)+1$. At least two nodes in $\{i+1, next(i)-1\}$ do not receive links.

Next suppose $next(i)$ is activated by $S$. If $next(i)$ was activated by $next(i)-1$, then all nodes in $\{i+1, \ldots, next(i)-1\}$ must have received links, which

contradicts either the minimality of $S$, or the assertion that $i$ gets a link in every optimal solution. So we conclude that either $next(i)$ received a link in $S$ or was activated by $next(i)+1$. Since $S$ is minimal, there must exist at least one node in $\{i+1,\ldots,next(i)-1\}$ that did not receive a link. If there is exactly one such node $p$, then clearly every node in $\{i,\ldots,next(i)\}$ is activated by $S$, but then solution $S'$ which moves the link given to $i$ to node $p$ also activates the same nodes, and contradicts the assertion that every optimal solution must give a link to $i$. Therefore, there must be two nodes in $\{i+1,\ldots,next(i)-1\}$ that do not receive links (see Fig. 5). Therefore $|S_1| \leq next(i)-i-3$. Consider now the link set $S'$ made by *shifting* all the links in $S_1$ from $i+1$ onwards to a consecutive sequence of nodes ending with $next(i)-1$, that is, $S' = \{i\} \cup \{next(i)-|S_1|,\ldots next(i)-1\} \cup S_2$. Then $S'$ is the same size as $S$, it activates exactly the same number of nodes as $S$, and is therefore also optimal. Observe that $next(i)-|S_1| > i+2$, since $|S_1| \leq next(i)-i-3$. Thus $S'$ is an optimal solution to $MIA(P_{i,j},k)$ that does not give links to $i+1$ and $i+2$ as needed. □

The following lemma summarizes the optimal substructure of the problem when $t(i)=1$.

**Lemma 8.** *Given a sub-path $P_{i,j}$ in which all nodes have threshold 1 or 2, and $t(i)=1$:*

$$A(i,j,k) = \begin{cases} min\{k, j-i+1\} & \text{if } next(i) > j \\ max \begin{cases} 1+A(i+1,j,k) \\ 1+B(i+1,j,k-1) \end{cases} & \text{if } next(i) \leq j \end{cases}$$

$$B(i,j,k) = \begin{cases} 0 & \text{if } \ next(i) > j \\ B(i+1,j,k) & \text{if } \ next(i) \leq j \end{cases}$$

*Proof.* First we prove the correctness of the recursive definitions for $A(i,j,k)$. If $next(i) > j$, then node $i$ must receive a link, as it is not possible otherwise to activate any node in $P_{i+1,j}$. Thus node $i$ receives a link and is activated first. Inductively, we can show that node $i+1$ is the second node which receives a link and gets activated (see Fig. 6). Therefore, all nodes in $\{i,\ldots,i+min\{k,j-i+1\}-1\}$ must receive a link. Any node that does not receive a link cannot get activated, the maximum number of activated nodes is $min\{k,n-i+1\}$. This shows that $A(i,j,k) = min\{k,n-i+1\}$ in this case.

Otherwise, $next(i) \leq j$, that is, node $i$ is not the rightmost node with threshold 1. First observe that either there exists an optimal solution for $MIA(P_{i,j},k)$ in which node $i$ does not receive a link, or in every optimal solution for $MIA(P_{i,j},k)$, node $i$ receives a link. In the first case, let $S$ be an optimal solution for $MIA(P_{i,j},k)$ in which node $i$ does not receive a link. It follows that its neighbor node $i+1$ was also activated. Clearly $S$ must be an optimal solution for $MIA(P_{i+1,j},k)$ (if not, and if $S'$ is a solution for $MIA(P_{i+1,j},k)$ that activates more nodes than $S$, then $S'$ is also a better solution for $MIA(P_{i,j},k)$, contradicting the optimality of $S$. Therefore, $A(i,k) = 1+A(i+1,k)$.

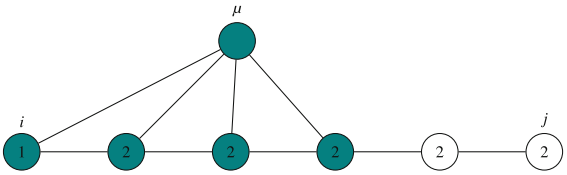



**Fig. 6.** An optimal link set $S$ for $MIA(P_{i,j}, 4)$: the case when $t(i) = 1$ and $next(i) > j$

In the second case, by Lemma 7, we have an optimal solution $S'$ in which nodes $i + 1$ and $i + 2$ do not receive links and are therefore not activated. Furthermore, using a cut-and-paste argument, it is straightforward to see that $S'' = S' - \{i\}$ is an optimal solution for $MIB(P_{i+1,j}, k - 1)$. It follows that $A(i, k) = 1 + B(i + 1, k - 1)$.

Finally, any solution in which node $i$ is not activated, we can be sure that neither node $i$ gets a link, nor does its neighbor, node $i+1$ get activated. Therefore $B(i, k) = B(i + 1, k)$. This completes the proof. □

We now use the definitions of $A(i, j, k)$ and $B(i, j, k)$ to prove the main result of this section:

**Theorem 3.** Max-Inf-Links *$(P_n, k)$ can be solved in time $\Theta(kn)$.*

*Proof.* If $P_n$ has no nodes of threshold $> 2$, then clearly $MI(P_n) = max\{A(1, n, k), B(1, n, k)\}$. If not, let $q$ be the largest index of a node in $P_n$ such that $t(q) \geq 3$. If $t(q) > 3$, then $q$ cannot be activated, therefore, there is no need to give a link to $q$. That is, the optimal solution to the Max-Inf-Links problem with $k$ links does not give node $q$ a link, and instead uses $\ell$ links to solve the Max-Influence problem on the path $P_{1,q-1}$ and $k - \ell$ links to solve the Max-Influence problem on the path $P_{q+1,n}$ for some value of $\ell$ such that $0 \leq \ell \leq k$. Thus the optimal solution can be found by checking for all possible values of $\ell$ between 0 and $k$, yielding:

$$MI(P_n, k) = max_{0 \leq \ell \leq k}\{MI(P_{1,q-1}, \ell) + MI(P_{q+1,n}, k - \ell)\}$$

Finally if $t(q) = 3$, then either there exists an optimal solution in which $q$ is not activated, in which case $MI(P_n, k)$ is defined identically to the case when $t(q) > 3$, or in every optimal solution $q$ is activated. In this case, let $S$ be an optimal solution to $MI(P_n, k)$. By assumption, $q$ is activated. Since $t(q) = 3$, it must be that $q$ gets a link, and also that $q - 1$ and $q + 1$ are activated. Let $S = S_1 \cup \{q\} \cup S_2$ where $S_1 = \{1, \ldots, q - 1\} \cap S$ and $S_2 = S \cap \{q + 1, n\}$, and let $|S_1| = \ell$ and $|S_2| = k - 1 - \ell$ We now claim that $S_2$ is an optimal solution to $MIA(P_{q+1,n}, k-1-\ell)$. If there was a better solution to $MIA(P_{q+1,n}, k-1-\ell)$ than $S_2$, we can replace $S_2$ by the claimed better solution in $S$ to get a better solution to Max-Inf-Links$(P_n, k)$, a contradiction to the optimality of $S$.

Next we claim that $S_1$ is an optimal solution to Max-Inf-Links$(P_{1,q-1}, \ell)$. Suppose instead that $S_1$ activates $\alpha$ nodes in $P_{1,q-1}$ and there is another set of

links $S'$ of size $\ell$ that activates $\beta > \alpha$ nodes in $P_{1,q-1}$. If $S'$ activates $q-1$, then clearly $S' \cup \{q\} \cup S_2$ activates more nodes than $S$ in $P_n$, a contradiction to the optimality of $S$. If $S'$ does not activate $q$, then $S' \cup S_2$ does not activate node $q$ but activates at least the same number of nodes in $P_n$ as $S$ does, a contradiction to the assertion that every optimal solution activates $q$. Therefore, we conclude that

$$MI(P_n, k) = max_{0 \leq \ell \leq k}\{1 + MI(P_{1,q-1}, \ell) + A(P_{q+1,n}, k-1-\ell)\}$$

Finally, we prove that the above formulation can be computed using dynamic programming in time $\Theta(kn)$. For any sub-path$P_{i,j}$ containing only nodes of threshold 1 and 2, the values of $A(i,j,r)$, $B(i,j,r)$ and $MI(i,j,r)$ can be found using the definitions in Lemmas 8 and 6 for all $0 \leq r \leq k$ in time $\Theta(k(j-i+1)$. Since the total lengths of all sub-paths is at most $n$, the total time spent is $\Theta(nk)$. Since there are $O(n)$ such sub-paths, the recursive formulation for $MI(P_n, k)$ above for a paths with no threshold limit takes another $O(kn)$ time to compute. □

### 5.1 Cycles

By taking out a single node from the cycle, and considering the resulting path, we can obtain an algorithm for the Max-Inf-Linksproblem for cycles:

**Theorem 4.** *The* Max-Inf-Links *problem for a cycle $C_n$ using $k$ links, can be solved in time $\theta(kn^2)$.*

## 6 $\Theta(n)$ Algorithm for Cliques

The following greedy algorithm can be shown to produce an optimal solution for cliques. We sort the nodes in order of threshold and examine nodes in order while we still have links to assign. When we process node $i$, if $t(i) > i$, we stop assigning links and break. If $t(i) < i$, we simply increment $i$ and continue. Finally if $t(i) = i$, we give a link to node $i$.

**Theorem 5.** Max-Inf-Links*$(K_n, k)$ can be solved in time $\Theta(n)$.*

## 7 Discussion

In this paper, we introduced and studied the Max-Influence-with-Links problem: given a social network $G$ where every node $v$ has a threshold $t(v)$ to be activated, and an integer $k$, which $k$ nodes in $G$ should an already activated external influencer $\mu$ befriend, so as to influence the maximum possible number of nodes in the network? We showed that the problem is NP-complete, even for a single influencer, and bipartite graphs with maximum threshold 2. In contrast, for the case of a single external influencer, we showed a linear time algorithm for cliques, $\Theta(kn)$ algorithm for paths, $\Theta(kn^2)$ algorithm for cycles, and a $\Theta(k^2n^2)$

algorithm for trees. It seems straightforward to generalize our algorithms for any number $k$ of external influencers. It would be interesting to study the complexity of the problem for graphs of bounded tree width, and the approximability of the problem for constant $k$. We are also interested in studying the case with non-uniform weights on the edges. Clearly, the problem remains NP-complete in general, but the complexity for special classes of graphs remains open.

## References


1. Ben-Zwi, O., Hermelin, D., Lokshtanov, D., Newman, I.: Treewidth governs the complexity of target set selection. Discrete Optim. **8**, 702–715 (2011)
2. Borgs, C., Brautbar, M., Chayes, J., Lucier, B.: Maximizing social influence in nearly optimal time. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, pp. 946–957 (2014)
3. Brown, J.J., Reingen, P.H.: Social ties and word-of-mouth referral behavior. J. Consum. Res. **14**, 350–362 (1987)
4. Chen, N.: On the approximability of influence in social networks. In: Proceedings of the Symposium on Discrete Algorithms, SODA 2008, pp. 1029–1037 (2008)
5. Chen, W., Wang, Y., Yang, S.: Efficient influence maximization in social networks. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009, pp. 199–208 (2009)
6. Cicalese, F., Cordasco, G., Gargano, L., Milanic, M., Peters, J.G., Vaccaro, U.: How to go viral: cheaply and quickly. In: Ferro, A., Luccio, F., Widmayer, P. (eds.) Fun with Algorithms. LNCS, vol. 8496, pp. 100–112. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07890-8_9
7. Cicalese, F., Cordasco, G., Gargano, L., Milanič, M., Vaccaro, U.: Latency-bounded target set selection in social networks. In: Bonizzoni, P., Brattka, V., Löwe, B. (eds.) CiE 2013. LNCS, vol. 7921, pp. 65–77. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39053-1_8
8. Cordasco, G., Gargano, L., Rescigno, A.A., Vaccaro, U.: Optimizing spread of influence in social networks via partial incentives. In: Scheideler, C. (ed.) Structural Information and Communication Complexity. LNCS, vol. 9439, pp. 119–134. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25258-2_9
9. de Caen, D.: An upper bound on the sum of squares of degrees in a graph. Discrete Math. **185**, 245–248 (1998)
10. Demaine, E.D., Hajiaghayi, M.T., Mahini, H., Malec, D.L., Raghavan, S., Sawant, A., Zadimoghadam, M.: How to influence people with partial incentives. In: Proceedings of the International Conference on World Wide Web, WWW 2014, pp. 937–948 (2014)
11. Dinh, T.N., Zhang, H., Nguyen, D.T., Thai, M.T.: Cost-effective viral marketing for time-critical campaigns in large-scale social networks. IEEE/ACM Trans. Netw. **22**, 2001–2011 (2014)
12. Domingos, P., Richardson, M.: Mining the network value of customers. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2001, pp. 57–66 (2001)
13. Eftekhar, M., Ganjali, Y., Koudas, N.: Information cascade at group scale. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, pp. 401–409 (2013)

14. Fazli, M.A., Ghodsi, M., Habibi, J., Jalaly Khalilabadi, P., Mirrokni, V., Sadeghabad, S.S.: On the non-progressive spread of influence through social networks. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 315–326. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29344-3_27
15. Gargano, L., Hell, P., Peters, J., Vaccaro, U.: Influence diffusion in social networks under time window constraints. In: Moscibroda, T., Rescigno, A.A. (eds.) SIROCCO 2013. LNCS, vol. 8179, pp. 141–152. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03578-9_12
16. Goldenberg, J., Libai, B., Muller, E.: Talk of the network: a complex systems look at the underlying process of word-of-mouth. Mark. Lett. **12**, 211–223 (2001)
17. Goyal, A., Bonchi, F., Lakshmanan, L.V.S.: A data-based approach to social influence maximization. Proc. VLDB Endow. **5**, 73–84 (2011)
18. Goyal, A., Bonchi, F., Lakshmanan, L.V.S., Venkatasubramanian, S.: On minimizing budget and time in influence propagation over social networks. Soc. Netw. Anal. Min. **3**, 179–192 (2013)
19. Goyal, A., Lu, W., Lakshmanan, L.V.S.: Celf++: optimizing the greedy algorithm for influence maximization in social networks. In: Proceedings of the International Conference Companion on World Wide Web, WWW 2011, pp. 47–48 (2011)
20. Gunnec, D., Raghavan, S.: Integrating social network effects in the share-of-choice problem. Technical report, University of Maryland, College Park (2012)
21. Gunnec, D., Raghavan, S., Zhang, R.: The least cost influence problem. Technical report, University of Maryland, College Park (2013)
22. He, J., Ji, S., Beyah, R., Cai, Z.: Minimum-sized influential node set selection for social networks under the independent cascade model. In: Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2014, pp. 93–102 (2014)
23. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2003, pp. 137–146 (2003)
24. Kempe, D., Kleinberg, J., Tardos, É.: Influential nodes in a diffusion model for social networks. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1127–1138. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_91
25. Lafond, M., Narayanan, L., Wu, K.: Whom to befriend to influence people. In: Suomela, J. (ed.) SIROCCO 2016. LNCS, vol. 9988, pp. 340–357. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48314-6_22
26. Lamba, H., Pfeffer, J.: Maximizing the spread of positive influence by deadline. In: Proceedings of the International Conference Companion on World Wide Web, WWW 2016, pp. 67–68 (2016)
27. Leskovec, J., Adamic, L.A., Huberman, B.A.: The dynamics of viral marketing. In: Proceedings of the ACM Conference on Electronic Commerce, pp. 228–237 (2006)
28. Lu, W., Bonchi, F., Goyal, A., Lakshmanan, L.V.S.: The bang for the buck: fair competitive viral marketing from the host perspective. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, pp. 928–936 (2013)
29. Lv, S., Pan, L.: Influence maximization in independent cascade model with limited propagation distance. In: Han, W., Huang, Z., Hu, C., Zhang, H., Guo, L. (eds.) APWeb 2014. LNCS, vol. 8710, pp. 23–34. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11119-3_3
30. Mossel, E., Roch, S.: Submodularity of influence in social networks: from local to global. SIAM J. Comput. **39**, 2176–2188 (2010)

31. Nguyen, H.T., Thai, M.T., Dinh, T.N.: Stop-and-stare: optimal sampling algorithms for viral marketing in billion-scale networks. In: Proceedings of the International Conference on Management of Data, SIGMOD 2016, pp. 695–710 (2016)
32. Nichterlein, A., Niedermeier, R., Uhlmann, J., Weller, M.: On tractable cases of target set selection. Soc. Netw. Anal. Min. **3**(2), 233–256 (2012)
33. Richardson, M., Domingos, P.: Mining knowledge-sharing sites for viral marketing. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2002, pp. 61–70 (2002)
34. Tang, S., Yuan, J.: Going viral: optimizing discount allocation in social networks for influence maximization (2016). CoRR abs/1606.07916
35. Tang, Y., Shi, Y., Xiao, X.: Influence maximization in near-linear time: a martingale approach. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2015, pp. 1539–1554 (2015)
36. Tang, Y., Xiao, X., Shi, Y.: Influence maximization: near-optimal time complexity meets practical efficiency. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2014, pp. 75–86 (2014)
37. Yang, Y., Mao, X., Pei, J., He, X.: Continuous influence maximization: what discounts should we offer to social network users? In: Proceedings of the International Conference on Management of Data, SIGMOD 2016, pp. 727–741 (2016)

# Effective Edge-Fault-Tolerant Single-Source Spanners via Best (or Good) Swap Edges

Davide Bilò[1], Feliciano Colella[2(✉)], Luciano Gualà[3], Stefano Leucci[4], and Guido Proietti[5,6]

[1] Dipartimento di Scienze Umanistiche e Sociali, University of Sassari, Sassari, Italy
davide.bilo@uniss.it
[2] Gran Sasso Science Institute, L'Aquila, Italy
feliciano.colella@gssi.it
[3] Dipartimento di Ingegneria dell'Impresa, University of Rome "Tor Vergata", Rome, Italy
guala@mat.uniroma2.it
[4] Department of Computer Science, ETH Zürich, Zürich, Switzerland
stefano.leucci@inf.ethz.ch
[5] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, University of L'Aquila, L'Aquila, Italy
guido.proietti@univaq.it
[6] Istituto di Analisi dei Sistemi ed Informatica, CNR, Rome, Italy

**Abstract.** Computing *all best swap edges* (ABSE) of a spanning tree $T$ of a given $n$-vertex and $m$-edge undirected and weighted graph $G$ means to select, for each edge $e$ of $T$, a corresponding non-tree edge $f$, in such a way that the tree obtained by replacing $e$ with $f$ enjoys some optimality criterion (which is naturally defined according to some objective function originally addressed by $T$). Solving efficiently an ABSE problem is by now a classic algorithmic issue, since it conveys a very successful way of coping with a (transient) *edge failure* in tree-based communication networks: just replace the failing edge with its respective swap edge, so as that the connectivity is promptly reestablished by minimizing the rerouting and set-up costs. In this paper, we solve the ABSE problem for the case in which $T$ is a *single-source shortest-path tree* of $G$, and our two selected swap criteria aim to minimize either the *maximum* or the *average stretch* in the swap tree of all the paths emanating from the source. Having these criteria in mind, the obtained structures can then be reviewed as *edge-fault-tolerant single-source spanners*. For them, we propose two efficient algorithms running in $O(mn + n^2 \log n)$ and $O(mn \log \alpha(m, n))$ time, respectively, and we show that the guaranteed (either maximum or average, respectively) stretch factor is equal to 3, and this is tight. Moreover, for the maximum stretch, we also propose an almost linear $O(m \log \alpha(m, n))$ time algorithm computing a set of *good* swap edges, each of which will guarantee a relative approximation factor on the maximum stretch of 3/2 (tight) as opposed to that provided by the corresponding BSE. Surprisingly, no previous results were known for these two very natural swap problems.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 303–317, 2017.
https://doi.org/10.1007/978-3-319-72050-0_18

## 1 Introduction

Nowadays there is an increasing demand for an *efficient* and *resilient* information exchange in communication networks. This means to design on one hand a logical structure onto a given communication infrastructure, which optimizes some sought routing protocol in the absence of failures, and on the other hand, to make such a structure resistant against possible link/node malfunctioning, by suitably adding to it a set of redundant links, which will enter into operation as soon as a failure takes place.

More formally, the depicted situation can be modeled as follows: the underlying communication network is an $n$-vertex and $m$-edge undirected input graph $G = (V(G), E(G), w)$, with positive real edge weights defined by $w$, the logical (or primary) structure is a (spanning) subgraph $H$ of $G$, and finally the additional links is a set of edges $A$ in $E(G) \setminus E(H)$. Under normal circumstances, communication takes place on $H$, by following a certain protocol, but as soon as an edge in $H$ fails, then one or more edges in $A$ come into play, and the communication protocol is suitably adjusted.

In particular, if the primary structure is a (spanning) *tree* of $G$, then a very effective way of defining the set of additional edges is the following: with each tree edge, say $e$, we associate a so-called *best swap edge*, namely a non-tree edge that will replace $e$ once it (transiently) fails, in such a way that the resulting *swap tree* enjoys some nice property in terms of the currently implemented communication protocol. By doing in this way, rerouting and set-up costs will be minimized, in general, and the quality of the post-failure service remains guaranteed. Then, an *all best swap edges* (ABSE) problem is that of finding efficiently (in term of time complexity) a best swap edge for each tree edge.

Due to their fault-tolerance application counterpart, ABSE problems received a large attention by the algorithmic community. In such a framework, a key role has been played by the *Shortest-Path Tree* (SPT) structure, which is commonly used for implementing efficiently the *broadcasting* communication primitive. Indeed, it is was shown already in [15] that an effective post-swap broadcast protocol can be put in place just after the original SPT undergoes an edge failure. Not surprisingly then, several ABSE problems w.r.t. an SPT have been studied in the literature, for many different swap criteria.

*Previous work on swapping in an SPT.* Since an SPT enjoys several optimality criteria when looking at distances from the source, say $s$, several papers have analyzed the problem in various respects. However, most of the efforts focused on the minimization w.r.t. the following two swap criterion: the maximum/average distance from $s$ to any node which remained disconnected from $s$ after a failure. The currently fastest solutions for these two ABSE problems run in $O(m \log \alpha(m, n))$ time [5] and $O(m\, \alpha(n, n) \log^2 n)$ time [8], respectively. Moreover, it has been shown that in the swap tree the maximum (resp., average) distance of the disconnected nodes from $s$ is at most twice (resp., triple) that of the new optimum SPT [18], and these bounds are tight.

Other interesting swap criteria which have been analyzed include the minimization of the maximum increase (before and after the failure) of the distance from $s$, and the minimization of the distance from $s$ to the root of the subtree that gets disconnected after the failure [17]. Besides the centralized setting, all these swap problems have been studied also in a distributed framework (e.g., see [7,10–12]).

On the other hand, no results are known for the case in which one is willing to select a BSE with the goal of minimizing either the *maximum* or the *average stretch* from the source $s$ of the disconnected nodes, where the stretch of a node is measured as the ratio between its distance from $s$ in the swap tree and in a new optimum SPT. This is very surprising, since they are (especially the former one) the universally accepted criterion leading to the design of a *spanner*, i.e., a sparse subgraph preserving shortest paths (between pairs of vertices of interest) in a graph (also in the presence of failures).

In this paper, we aim to fill this gap, by providing efficient solutions exactly for these two swap criteria.

*Our results.* Let us denote by `ABSE-MS` and `ABSE-AS` the ABSE problem w.r.t. the maximum and the average stretch swap criterion, respectively. For such problems, we devise two efficient algorithms running in $O(mn + n^2 \log n)$ and $O(mn \log \alpha(m, n))$ time, respectively. Notice that both solutions incorporate the running time for computing all the replacement shortest paths from the source after the failure of every edge of the SPT, as provided in [13], whose computation essentially dominates in an asymptotic sense the time complexity. Our two solutions are based on independent ideas, as described in the following:

- for the `ABSE-MS` problem, we develop a *centroid decomposition* of the SPT, and we exploit a distance property that has to be enjoyed by a BSE w.r.t. a nested and log-depth hierarchy of centroids, which will be defined by the subtree detached from the source after the currently analyzed edge failure. A further simple filtering trick on the set of potential swap edges will allow to reduce them from $O(m)$ to $O(n)$, thus returning the promised $O(n^2 \log n)$ time.
- for the `ABSE-AS` problem, we instead suitably combine a set of linearly-computable (at every edge fault) information, that essentially will allow to describe in $O(1)$ time the quality of a swap edge. This procedure is in principle not obvious, since to compute the average stretch we need to know, for each swap edge, the $O(n)$ distances to all the nodes in the detached subtree. Again, by filtering on the set of potential swap edges, we will get an $O(n^2)$ running time, which will be absorbed by the all-replacement paths time complexity.

Concerning the quality of the corresponding swap trees, we instead show that the guaranteed (either maximum or average, respectively) stretch factor w.r.t. the paths emanating from the source (in the surviving graph) is equal to 3, and this is tight. By using a different terminology, our structures can then be revised as *edge-fault-tolerant single-source* 3*-spanners*, and we qualified them

as *effective* since they can be computed quickly, are very sparse, provide a very simple alternative post-failure routing, and finally have a small (either maximum or average) stretch.

Although the proposed solutions are quite efficient, their running time can become prohibitive for large and dense input graphs, since in this case they would amount to a time cubic in the number of vertices. Unfortunately, it turns out that their improvement is unlikely to be achieved, unless one could avoid the explicit recomputation of all post-failure distances from the source. To circumvent this problem, we then adopt a different approach, which by the way finds application for the (most relevant) max-stretch measure only: we renounce to optimality in the detection of a BSE, in return of a substantial improvement (in the order of a linear factor in $n$) in the runtime. More precisely, for such a measure, we will compute in an almost linear $O(m \log \alpha(m, n))$ time a set of *good* swap edges (GSE), each of which will guarantee a relative approximation factor on the maximum stretch of 3/2 (tight) as opposed to that provided by the corresponding BSE. Moreover, a GSE will still guarantee an absolute maximum stretch factor w.r.t. the paths emanating from the source (in the surviving graph) equal to 3 (tight).

Besides that, we also point out another important feature concerned with the computation in a *distributed* setting of all our good swap edges. Indeed, in [7] it was shown that they can be computed in an *asynchronous message passing system* in essentially optimal *ideal time*,[1] space usage, and message complexity, as opposed to the recomputation of all the corresponding BSE, for which no efficient solution is currently available.

*Other related results.* Besides swap-based approaches, an SPT can be made edge-fault-tolerant by further enriching the set of additional edges, so that the obtained structure has almost-shortest paths emanating from the source, once an edge fails. The currently best trade off between the size of the set of additional edges and the quality of the resulting paths emanating from $s$ is provided in [3], where the authors showed that for any arbitrary constant $\varepsilon > 0$, one can compute in polynomial time a slightly superlinear (in $n$, and depending on $\varepsilon$) number of additional edges in such a way that the resulting structure retains $(1 + \varepsilon)$-stretched post-failure paths from the source.

For the sake of completeness, we also quickly recall the main results concerned with ABSE problems. For the *minimum spanning tree* (MST), a BSE is of course one minimizing the *cost* of the swap tree, i.e., a swap edge of minimum cost. This problem is also known as the MST *sensitivity analysis* problem, and can be solved in $O(m \log \alpha(m, n))$ time [19]. Concerning the *minimum diameter spanning tree*, a BSE is instead one minimizing the *diameter* of the swap tree [14,17], and the best solution runs in $O(m \log \alpha(m, n))$ time [5]. Regarding the *minimum routing-cost spanning tree*, a BSE is clearly one minimizing the *all-to-all routing*

[1] This is the time obtained with the ideal assumption that the communication time of each message to a neighboring process takes constant time, as in the synchronous model.

*cost* of the swap tree [21], and the fastest solutions for solving this problem has a running time of $O\left(m2^{O(\alpha(n,n))}\log^2 n\right)$ [4]. Finally, for a *tree spanner*, a BSE is one minimizing the maximum stretch w.r.t. the all pair distances, and the fastest solution to date run in $O(m^2 \log \alpha(m,n))$ time [1].

To conclude, we point out that the general problem of designing fault-tolerant spanners for the *all-to-all* case has been extensively studied in the literature, and we refer the interested reader to [2,6,9] and the references therein.

## 2 Problem Definition

Let $G = (V(G), E(G), w)$ be a 2-edge-connected, edge-weighted, and undirected graph with cost function $w : E(G) \to \mathbb{R}^+$. We denote by $n$ and $m$ the number of vertices and edges of $G$, respectively. If $X \subseteq V$, let $E(X)$ be the set of edges incident to at least one vertex in $X$. Given an edge $e \in E(G)$, we will denote by $G - e$ the graph obtained from $G$ by removing edge $e$. Similarly, given a vertex $v \in V(G)$, we will denote by $G - v$ the graph obtained from $G$ by removing vertex $v$ and all its incident edges. Let $T$ be an SPT of $G$ rooted at $s \in V(G)$. Given an edge $e \in E(T)$, we let $C(e)$ be the set of all the *swap edges* for $e$, i.e., all edges in $E(G) \setminus \{e\}$ whose endpoints lie in two different connected components of $T - e$, and let $C(e, X)$ be the set of all the swap edge for $e$ incident to a vertex in $X \subseteq V(G)$. For any $e \in E(T)$ and $f \in C(e)$, let $T_{e/f}$ denote the *swap tree* obtained from $T$ by replacing $e$ with $f$. Let $T_v = (V(T_v), E(T_v))$ be the subtree of $T$ rooted at $v \in V(G)$. Given a pair of vertices $u, v \in V(G)$, we denote by $d_G(u, v)$ the *distance* between $u$ and $v$ in $G$. Moreover, for a swap edge $f = (x, y)$, we assume that the first appearing endvertex is the one closest to the source, and we may denote by $w(x, y)$ its weight. We define the *stretch factor of $y$ w.r.t.* $s, T, G$ as $\sigma_G(T, y) = \frac{d_T(s,y)}{d_G(s,y)}$.

Given an SPT $T$ of $G$, the `ABSE-MS` problem is that of finding, for each edge $e = (a, b) \in E(T)$, a swap edge $f^*$ such that:

$$f^* \in \arg\min_{f \in C(e)} \left\{ \mu(f) := \max_{v \in V(T_b)} \sigma_{G-e}\big(T_{e/f}, v\big) \right\}.$$

Similarly, the `ABSE-AS` problem is that of finding, for each edge $e = (a, b) \in E(T)$, a swap edge $f^*$ such that:

$$f^* \in \arg\min_{f \in C(e)} \left\{ \lambda(f) := \frac{1}{|V(T_b)|} \sum_{v \in V(T_b)} \sigma_{G-e}\big(T_{e/f}, v\big) \right\}.$$

We will call $\mu(f)$ (resp., $\lambda(f)$) the *max*-(resp., *avg*-)*stretch* of $f$ w.r.t. $e$.

## 3 An Algorithm for `ABSE-MS`

In this section we will show an efficient algorithm to solve the `ABSE-MS` problem in $O(mn + n^2 \log n)$ time. Notice that a brute-force approach would require

$O(mn^2)$ time, given by the $O(n)$ time which is needed to evaluate the quality of each of the $O(m)$ swap edges, for each of the $n-1$ edges of $T$. Our algorithm will run through $n-1$ phases, each returning in $O(m+n\log n)$ time a BSE for a failing edge of $T$, as described in the following.

Let us fix $e=(a,b)$ as the failing edge. First, we compute in $O(m+n\log n)$ time all the distances in $G-e$ from $s$. Then, we filter the $O(m)$ potential swap edges to $O(n)$, i.e., at most one for each node $v$ in $T_b$. Such a filtering is simply obtained by selecting, out of all edges $f=(x,v)\in C(e,\{v\})$, the one minimizing the measure $d_G(s,x)+w(f)$. Indeed, it is easy to see that the max-stretch of such selected swap edge is never worse than that of every other swap edge in $C(e)$. This filtering phase will cost $O(m)$ total time. As a consequence, we will henceforth assume that $|C(e)|=O(n)$.

Then, out of the obtained $O(n)$ swap edges for $e$, we further restrict our attention to a subset of $O(\log n)$ *candidates* as BSE, which are computed as follows. Let $\Lambda$ denote a generic subtree of $T_b$, and assume that initially $\Lambda=T_b$. First of all, we compute in $O(|V(\Lambda)|)$ time a *centroid* $c$ of $\Lambda$, namely a node whose removal from $\Lambda$ splits $\Lambda$ in a forest $F$ of subtrees, each having at most $|V(\Lambda)|/2$ nodes [16]; then, out of all the swap edges, we select a candidate edge $f$ minimizing the distance from $s$ to $c$ in $T_{e/f}$, i.e.,

$$f\in\arg\min_{(x',v')\in C(e)}\left\{d_T(s,x')+w(x',v')+d_T(v',c)\right\};$$

then, we compute a *critical node* $z$ for the selected swap edge $f$, i.e.,

$$z\in\arg\max_{z'\in V(T_b)}\sigma_{G-e}\big(T_{e/f},z'\big).$$

We now select a suitable subtree $\Lambda'$ of the forest $F$, and we pass to the selection of the next candidate BSE by recursing on $\Lambda'$, until $|V(\Lambda')|=1$. More precisely, $\Lambda'$ is the first tree of $F$ containing the first vertex of $V(\Lambda)$ that is encountered by following the path in $T$ from $z$ towards $c$ (see Fig. 1).

Due to the property of the centroid, the number of recursions will be $O\big(\log|V(T_b)|\big)=O(\log n)$, as promised, each costing $O(n)$ time. Moreover, at least one of the candidate edges will be a BSE for $e$, and hence it suffices to choose the edge minimizing the maximum stretch among the corresponding $O(\log n)$ candidate edges. This step is done within the recursive procedure by comparing the current candidate edge $f$ with the best candidate resulting from the nested recursive calls.

A more formal description of each phase is shown in Algorithm 1. In the following we prove the correctness of our algorithm.

**Lemma 1.** *Let $e=(a,b)$ be a failing edge, and let $\Lambda$ be a subtree of $T_b$. Given a vertex $c\in V(\Lambda)$, let $f\in\arg\min_{(x',v')\in C(e)}\left\{d_T(s,v')+w(x',v')+d_T(v',c)\right\}$ and let $z$ be a critical node for $f$. Let $F$ be the forest obtained by removing the edges incident to $c$ from $\Lambda$, and let $\Lambda'$ be the tree of $F$ containing the first vertex of the path from $z$ to $c$ in $T$ that is also in $V(\Lambda)$. For any swap edge $f'\in C(e,V(\Lambda))$, if $\mu(f')<\mu(f)$ then $f'\in C(e,V(\Lambda'))$.*

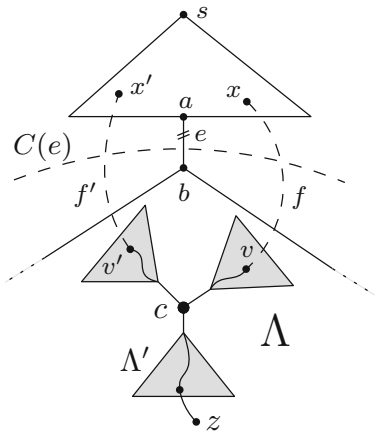



**Fig. 1.** The situation illustrated in Lemma 1. The subtree $\Lambda$ is represented by the three gray triangles along with the vertex c. $f = (x, v)$ is the candidate swap edge for $e$ that minimizes $d_T(s, x) + w(f) + d_T(v, c)$, and $z$ is its corresponding critical node. The algorithm will compute the next candidate swap edge by recursing on $\Lambda'$.

*Proof.* Let $f = (x, v)$ and $f' = (x', v')$. We show that if $v' \in V(\Lambda) \setminus V(\Lambda')$ then $\mu(f') \geq \mu(f)$ (see also Fig. 1). Indeed:

$$\begin{aligned}\mu(f') &\geq \sigma_{G-e}\big(T_{e/f'}, z\big) = \frac{d_{T_{e/f'}}(s,z)}{d_{G-e}(s,z)} = \frac{d_T(s,x') + w(f') + d_T(v',c) + d_T(c,z)}{d_{G-e}(s,z)} \\ &\geq \frac{d_T(s,x) + w(f) + d_T(v,c) + d_T(c,z)}{d_{G-e}(s,z)} \geq \frac{d_{T_e/f}(s,z)}{d_{G-e}(s,z)} = \sigma_{G-e}\big(T_{e/f}, z\big) \\ &= \mu(f),\end{aligned}$$

where we used the equality $d_T(v', z) = d_T(v', c) + d_T(c, z)$, which follows from the fact that the path from $v'$ to $z$ in $T$ must traverse $c$ as $v'$ and $z$ are in two different trees of $F$. □

**Lemma 2.** *If $C\big(e, V(\Lambda)\big)$ contains a BSE for $e$ then* `ABSE-MS`$(e, \Lambda)$ *returns a BSE for $e$.*

*Proof.* First of all notice that Algorithm 1 only returns edges in $C(e)$.

We prove the claim by induction on $|V(\Lambda)|$. If $|V(\Lambda)| = 1$ and $C\big(e, V(\Lambda)\big)$ contains a BSE $f^*$ for $e$, then let $f$ be the edge of $C(e)$ returned by Algorithm 1 and let $V(\Lambda) = \{c\}$. By choice of $f$, for every $v \in V(T_b)$,

$$d_{T_{e/f}}(s, v) \leq d_{T_{e/f}}(s, c) + d_T(c, v) = d_{T_{e/f^*}}(s, c) + d_T(c, v) = d_{T_{e/f^*}}(s, v),$$

from which we derive that $\mu(f) = \mu(f^*)$, and the claim follows.

If $|V(\Lambda)| > 1$ and $C\big(e, V(\Lambda)\big)$ contains a BSE for $e$, we distinguish two cases depending on whether the edge $f$ computed by Algorithm 1 is a BSE for $e$

**Algorithm 1.** ABSE-MS$(e, \Lambda)$

**Input** : a failing edge $e = (a, b) \in E(T)$, a subtree $\Lambda$ of $T_b$.
**Output**: an edge $f \in C(e)$. If $C\big(e, V(\Lambda)\big)$ contains a BSE for $e$, $f$ is a BSE for $e$.

```
1 c ← Centroid of Λ;
2 Let f ∈ arg min_{(x',v')∈C(e)} {d_T(s,x') + w(x',v') + d_T(v',c)};
3 if |V(Λ)| = 1 then return f;
4 Let z ∈ arg max_{z'∈V(T_b)} σ_{G−e}(T_{e/f}, z');
5 Let F be the forest obtained by removing the edges incident to c from Λ;
6 y ← first vertex along the path from z towards c in T that is also in V(Λ);
7 Λ' ← tree of F containing y;
8 f' ← ABSE-MS(e, Λ');
9 if μ(f') < μ(f) then return f' else return f;
```

or not. If that is the case, then $\mu(f) \leq \mu(f'') \, \forall f'' \in C(e)$ and the algorithm correctly returns $f$. Otherwise, by Lemma 1, any edge $f' \in C(e, V(\Lambda))$ such that $\mu(f') < \mu(f)$ must belong to $C\big(e, V(\Lambda')\big)$. It follows that $\Lambda'$ contains a BSE for $e$ and since $1 \leq |V(\Lambda')| < |V(\Lambda)|$ we have, by inductive hypothesis, that the edge $f'$ returned by ABSE-MS$(G, e, \Lambda')$ is a BSE for $e$. Clearly $\mu(f') < \mu(f)$ and hence Algorithm 1 correctly returns $f'$. □

Since each invocation of Algorithm 1 requires $O(n)$ time, Lemma 2 together with the previous discussions allows us to state the main theorem of this section:

**Theorem 1.** *There exists an algorithm that solves the ABSE-MS problem in* $O(mn + n^2 \log n)$ *time.*

## 4 An Algorithm for ABSE-AS

In this section we show how the ABSE-AS problem can be solved efficiently in $O(mn \log \alpha(m, n))$ time. Our approach first of all, in a preprocessing phase, computes in $O(m\, n \log \alpha(m, n))$ time all the replacement shortest paths from the source after the failure of every edge of $T$ [13]. Then, the algorithm will run through $n - 1$ phases, each returning in $O(m)$ time a BSE for a failing edge of $T$, as described in the following. Thus, the overall time complexity will be dominated by the preprocessing step.

Let us fix $e = (a, b) \in E(T)$ as the failing edge of $T$. The idea is to show that, after a $O(n)$ preprocessing time, we can compute the avg-stretch $\lambda(f)$ of any $f$ in constant time. This immediately implies that we can compute a BSE for $e$ by looking at all $O(m)$ swap edges for $e$.

Let $U = V(T_b)$ and let $y$ be a node in $U$, we define:

$$M(y) = \sum_{v \in U} \frac{d_T(y, v)}{d_{G-e}(s, v)}$$

and

$$Q = \sum_{v \in U} \frac{1}{d_{G-e}(s,v)}.$$

Let $f = (x, y)$ be a candidate swap edge incident in $y \in U$. The avg-stretch of $f$ can be rewritten as:

$$\lambda(f) = \sum_{v \in U} \frac{d_T(s,x) + w(f) + d_T(y,v)}{d_{G-e}(s,v)} = \big(d_T(s,x) + w(f)\big)Q + M(y).$$

Hence, the avg-stretch of $f$ can be computed in $O(1)$ time, once $Q$ and $M(y)$ are available in constant time. Observe that $Q$ does not depend on $y$ and can be computed in $O(n)$ time. The rest of this section is devoted to show how to compute $M(y)$ for every $y \in U$ in $O(n)$ overall time.

**Computing $M(y)$ for all $y \in U$.** Let $y$ e $y'$ be two nodes in $U$ such that $y$ is a child of $y'$ in $T$. Moreover, let $U_y = V(T_y)$, and let $Q_y = \sum_{v \in U_y} \frac{1}{d_{G-e}(s,y)}$. Hence, we can rewrite $M(y)$ and $M(y')$ as follows:

$$M(y) = \sum_{v \in U_y} \frac{d_T(y,v)}{d_{G-e}(s,v)} + \sum_{v \in U - U_y} \frac{w(y,y') + d_T(y',v)}{d_{G-e}(s,v)}$$

and

$$M(y') = \sum_{v \in U_y} \frac{w(y,y') + d_T(y,v)}{d_{G-e}(s,v)} + \sum_{v \in U - U_y} \frac{d_T(y',v)}{d_{G-e}(s,v)}.$$

Therefore, we have:

$$M(y) = M(y') + w(y,y')\big(-Q_y + (Q - Q_y)\big) = M(y') + w(y,y')\big(Q - 2Q_y\big). \quad (1)$$

The above equation implies that $M(y)$ can be computed in $O(1)$ time, once we have computed $M(y')$, $Q$ and $Q_y$. As a consequence, we can compute all the $M(y)$'s as follows. First, we compute $Q_y$ for every $y \in D$ in $O(n)$ overall time by means of a postorder visit of $T_b$. Notice also that $Q = Q_b$. Then, we compute $M(b)$ explicitly in $O(n)$ time. Finally, we compute all the other $M(y)$'s by performing a preorder visit of $T_b$. When we visit a node $y$, we compute $M(y)$ in constant time using (1). Thus, the visit will take $O(n)$ time. We have proved the following:

**Theorem 2.** *There exists an algorithm that solves the* `ABSE-AS` *problem in* $O(mn \log \alpha(m,n))$ *time.*

## 5 An Approximate Solution for `ABSE-MS`

In this section we show that for the max-stretch measure we can compute in an almost linear $O(m \log \alpha(m,n))$ time, a set of *good* swap edges (GSE), each

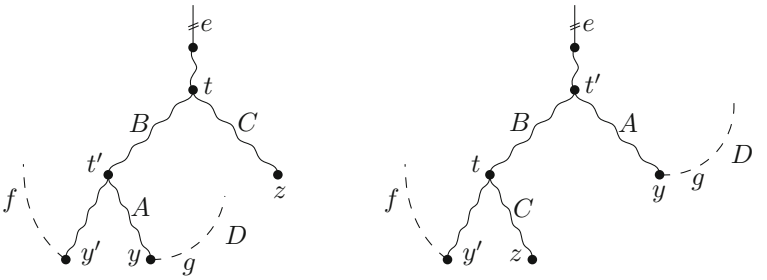



**Fig. 2.** The figure shows the two cases of the analysis, on the left $t$ is an ancestor of $t'$, while on the right the opposite holds. The splines denote a path, while the straight lines represent a single edge.

of which guarantees a relative approximation factor on the maximum stretch of 3/2 (tight), as opposed to that provided by the corresponding BSE. Moreover, as shown in the next section, each GSE still guarantees an absolute maximum stretch factor w.r.t. the paths emanating from the source (in the surviving graph) equal to 3 (tight).

**Lemma 3.** *Let $e$ be a failing edge in $T$, let*

$$g = (x, y) \in \arg \min_{(x',v') \in C(e)} \left\{ d_T(s, x') + w(x', v') \right\},$$

*and, finally, let $f = (x', y')$ be a best swap edge for $e$ w.r.t.* `ABSE-MS`*. Then, $\mu(g)/\mu(f) \leq 3/2$.*

*Proof.* Let $z$ be the critical node for the good swap edge $g$, and let $t$ (resp., $t'$) denote the *least common ancestor* in $T$ between $y'$ and $z$ (resp., $y'$ and $y$). Let $D = d_T(s, x) + w(x, y) = d_{G-e}(s, y)$. By choice of $g$, it holds that $d_{G-e}(s, z) \geq D$ and $d_{G-e}(s, y') \geq D$. We divide the proof into the following two cases, as depicted in Fig. 2: either (1) $t$ is an ancestor of $t'$ in $T$, or (2) $t'$ is an ancestor of $t$ in $T$. Let $A, B, C$ denote the distance in $T$ between $y$ and $t'$, $t'$ and $t$, $t$ and $z$, respectively.

**Case 1.** Since $t$ is an ancestor of $t'$ (left side of Fig. 2), we have that $d_{T_{e/f}}(s, y) \geq D + A$ and we can write:

$$\sigma_{G-e}(T_{e/f}, y) \geq \frac{D + A}{d_{G-e}(s, y)} = \frac{D + A}{D} \geq \frac{D + A}{d_{G-e}(s, z)},$$

and similarly $\sigma_{G-e}(T_{e/f}, z) \geq \frac{D+B+C}{d_{G-e}(s,z)}$. Moreover, by the definition of $\mu(\cdot)$ we have that $\mu(f) \geq \max\{\sigma_{G-e}(T_{e/f}, y), \sigma_{G-e}(T_{e/f}, z)\}$. The previous inequalities together imply:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{\sigma_{G-e}(T_{e/g}, z)}{\max\{\sigma_{G-e}(T_{e/f}, y), \sigma_{G-e}(T_{e/f}, z)\}} \leq \frac{A + B + C + D}{D + \max\{A, B + C\}}. \tag{2}$$

Now we divide the proof into two subcases, depending on whether $B+C \geq A$ or $B+C < A$. Observe that $D \geq d_G(s,y) \geq A$. If $B+C \geq A$, then (2) becomes:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{A+B+C+D}{B+C+D} = 1 + \frac{A}{B+C+D} \leq 1 + \frac{A}{2A} = \frac{3}{2},$$

otherwise, if $B+C < A$, then (2) becomes:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{A+B+C+D}{A+D} < \frac{2A+D}{A+D} = 1 + \frac{A}{A+D} \leq 1 + \frac{A}{2A} = \frac{3}{2}.$$

**Case 2.** Assume now that $t'$ is an ancestor of $t$ (right side of Fig. 2). Since

$$\mu(f) \geq \sigma_{G-e}(T_{e/f}, y) \geq \frac{d_{G-e}(s,y') + A + B}{d_{G-e}(s,y)} = \frac{d_{G-e}(s,y') + A + B}{D},$$

we have that:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq \frac{A+B+C+D}{d_{G-e}(s,z)} \cdot \frac{D}{d_{G-e}(s,y') + A + B} \\ &\leq \frac{A+B+C+D}{d_{G-e}(s,z)} \cdot \frac{D}{A+B+D}\end{aligned}$$

and since $d_{G-e}(s,z) \geq d_G(s,z) \geq C$, and recalling that $d_{G-e}(s,z) \geq D$, we have:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{A+B+C+D}{A+B+D} \cdot \frac{D}{\max\{C,D\}} = \left(1 + \frac{C}{A+B+D}\right) \cdot \frac{D}{\max\{C,D\}}. \quad (3)$$

Moreover, notice that also the following holds:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq \frac{\mu(g)}{\sigma_{G-e}(T_{e/f}, z)} \leq \frac{A+B+C+D}{d_{G-e}(s,z)} \cdot \frac{d_{G-e}(s,z)}{d_{G-e}(s,y') + d_T(y',t) + C} \\ &\leq \frac{A+B+C+D}{C+D} = 1 + \frac{A+B}{C+D}. \quad (4)\end{aligned}$$

We divide the proof into the following two subcases, depending on whether $D \geq C$ or $D < C$. In the first subcase, i.e., $D \geq C$, we have that (3) becomes $\frac{\mu(g)}{\mu(f)} \leq 1 + \frac{C}{A+B+D}$, and hence, by combining this inequality with (4), we obtain:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq 1 + \min\left\{\frac{C}{A+B+D}, \frac{A+B}{C+D}\right\} \\ &\leq 1 + \min\left\{\frac{C}{A+B+C}, \frac{A+B}{2C}\right\} \leq 1 + \frac{1}{2} = \frac{3}{2}.\end{aligned}$$

In the second subcase, i.e., $D < C$, (3) becomes:

$$\frac{\mu(g)}{\mu(f)} \leq \left(1 + \frac{C}{A+B+D}\right) \cdot \frac{D}{C} \leq \frac{D}{C} + \frac{D}{A+B+D} < 1 + \frac{D}{A+B+D}, \quad (5)$$

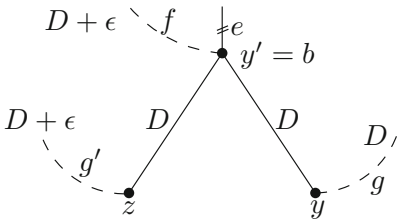



**Fig. 3.** A tight example showing that the quality of the good swap edge $g$ computed by the algorithm is a factor of 3/2 away from the qualify of a best swap edge $f$. In the picture, it is assumed that the distance from $s$ to $b$ is equal to 0, while the three dashed edges are assumed to be incident to the source, and the labels correspond to their weight. Then, $\mu(f) = \sigma_{G-e}(T_{e/f}, y) = \frac{2D+\epsilon}{D} \simeq 2$, while $\mu(g) = \sigma_{G-e}(T_{e/g}, z) = \frac{3D}{D+\epsilon} \simeq 3$, for small values of $\epsilon$.

and hence, by combining (5) and (4), we have that:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq 1 + \min\left\{\frac{D}{A+B+D}, \frac{A+B}{C+D}\right\} \\ &\leq 1 + \min\left\{\frac{D}{A+B+D}, \frac{A+B}{2D}\right\} \leq 1 + \frac{1}{2} = \frac{3}{2},\end{aligned}$$

from which the claim follows. □

Given the result of Lemma 3, we can derive an efficient algorithm to compute all the GSE for ABSE-MS. More precisely, in [18] it was shown how to find them in $O(m\,\alpha(m,n))$ time. Essentially, the approach used in [18] was based on a reduction to the *SPT sensitivity analysis* problem [20]. However, in [19] it was proposed a faster solution to such a problem, running in $O(m \log \alpha(m,n))$ time. Thus, we can provide the following

**Theorem 3.** *There exists a 3/2-approximation algorithm that solves the ABSE-MS problem in $O(m \log \alpha(m,n))$ time.*

We conclude this section with a tight example which shows that the analysis provided in Lemma 3 is tight (see Fig. 3).

## 6 Quality Analysis

As for previous studies on swap edges, it is interesting now to see how the tree obtained from swapping a failing edge $e = (a,b)$ with its BSE $f$ compares with a true SPT of $G - e$. According to our swap criteria, we will then analyze the lower and upper bounds of the max- and avg-stretch of $f$, i.e., $\mu(f)$ and $\lambda(f)$, respectively.

As already observed in the introduction, it is well-known [18] that for the swap edge, say $g$, which belongs to the shortest path in $G-e$ between $s$ and the

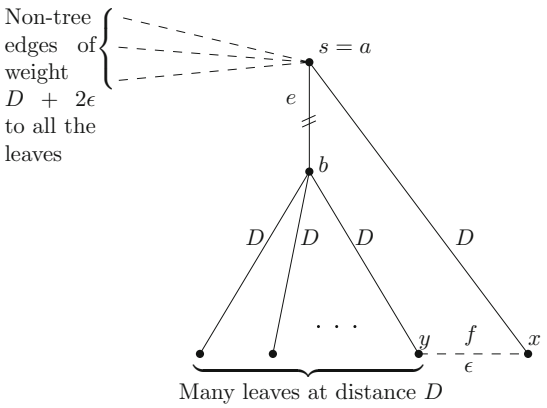



**Fig. 4.** Tight ratios for $\mu(f)$ and $\lambda(f)$. In the picture, the SPT $T$ (solid edges) along with the removed edge $e = (a, b)$ of weight 0; non-tree edges are dashed and the best swap edge (for both the maximum and the average stretch) is easily seen to be $f = (x, y)$, of weight $\epsilon > 0$. Then, we have that $\mu(f) = \frac{3D+\epsilon}{D+2\epsilon}$, which tends to 3 for small values of $\epsilon$, while $\lambda(f)$ tends to 3 as well, as soon as the number of leaves grows. Notice that $f$ is also a good swap edge for $e$.

root of the detached subtree $T_b$, we have that for any $v \in V(T_b)$, $\sigma_{G-e}(T_{e/g}, v) \leq 3$. This immediately implies that $\mu(g), \lambda(g) \leq 3$, namely $\mu(f), \lambda(f) \leq 3$. These bounds happen to be tight, as shown in Fig. 4.

Let us now analyze the lower and upper bounds of the max-stretch of a *good* swap edge $g$, i.e., $\mu(g)$, as defined in the previous section. First of all, once again it was proven in [17] that for any $v \in V(T_b)$, $\sigma_{G-e}(T_{e/g}, v) \leq 3$, which implies that $\mu(g) \leq 3$. Moreover, the example shown in Fig. 4 can be used to verify that this bound is tight.

## 7 Conclusions

In this paper we have studied two natural SPT swap problems, aiming to minimize, after the failure of any edge of the given SPT, either the maximum or the average stretch factor induced by a swap edge. We have first proposed two efficient algorithms to solve both problems. Then, aiming to the design of faster algorithms, we developed for the maximum-stretch measure an almost linear algorithm guaranteeing a 3/2-approximation w.r.t. the optimum.

Concerning future research directions, the most important open problem remains that of finding a linear-size edge-fault-tolerant SPT with a (maximum) stretch factor w.r.t. the root better than 3, or to prove that this is unfeasible. Another interesting open problem is that of improving the running time of our exact solutions. Notice that both our exact algorithms pass through the computation of all the post-failure single-source distances, and if we could avoid that we would get faster solutions. At a first glance, this sounds very hard, since

the stretches are heavily dependant on post-failure distances, but, at least in principle, one could exploit some monotonicity property among swap edges that could allow to skip such a bottleneck. Besides that, it would be nice to design a fast approximation algorithm for the average-stretch measure. Apparently, in this case it is not easy to adopt an approach based on good swap edges as for the maximum-stretch case, since swap edges optimizing other reasonable swap criteria (e.g., minimizing the distance towards the root of the detached subtree, or minimizing the distance towards a detached node) are easily seen to produce an approximation ratio of 3 as opposed to a BSE. A candidate solution may be that of selecting a BSE w.r.t. the sum-of-distances criterium, which can be solved in almost linear time [8], but for which we are currently unable to provide a corresponding comparative analysis.

Finally, we mention that a concrete task which will be pursued is that of conducting an extensive experimental analysis of the true performances of our algorithms, to check whether for real-world instances the obtained stretches are sensibly better or not w.r.t. the theoretical bounds.

## References

1. Bilò, D., Colella, F., Gualà, L., Leucci, S., Proietti, G.: A faster computation of all the best swap edges of a tree spanner. In: Scheideler, C. (ed.) Structural Information and Communication Complexity. LNCS, vol. 9439, pp. 239–253. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25258-2_17
2. Bilò, D., Grandoni, F., Gualà, L., Leucci, S., Proietti, G.: Improved purely additive fault-tolerant spanners. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 167–178. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_15
3. Bilò, D., Gualà, L., Leucci, S., Proietti, G.: Fault-tolerant approximate shortest-path trees. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 137–148. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44777-2_12
4. Bilò, D., Gualà, L., Proietti, G.: Finding best swap edges minimizing the routing cost of a spanning tree. Algorithmica **68**(2), 337–357 (2014)
5. Bilò, D., Gualà, L., Proietti, G.: A faster computation of all the best swap edges of a shortest paths tree. Algorithmica **73**(3), 547–570 (2015)
6. Chechik, S., Langberg, M., Peleg, D., Roditty, L.: Fault-tolerant spanners for general graphs. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, 31 May–2 June 2009, pp. 435–444 (2009)
7. Datta, A.K., Larmore, L.L., Pagli, L., Prencipe, G.: Linear time distributed swap edge algorithms. In: Spirakis, P.G., Serna, M. (eds.) CIAC 2013. LNCS, vol. 7878, pp. 122–133. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38233-8_11
8. Di Salvo, A., Proietti, G.: Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor. Theor. Comput. Sci. **383**(1), 23–33 (2007)
9. Dinitz, M., Krauthgamer, R.: Fault-tolerant spanners: better and simpler. In: Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, 6–8 June 2011, pp. 169–178 (2011)

10. Flocchini, P., Enriques, A.M., Pagli, L., Prencipe, G., Santoro, N.: Efficient protocols for computing the optimal swap edges of a shortest path tree. In: Levy, J.-J., Mayr, E.W., Mitchell, J.C. (eds.) TCS 2004. IIFIP, vol. 155, pp. 153–166. Springer, Boston, MA (2004). https://doi.org/10.1007/1-4020-8141-3_14
11. Flocchini, P., Enriques, A.M., Pagli, L., Prencipe, G., Santoro, N.: Point-of-failure shortest-path rerouting: computing the optimal swap edges distributively. IEICE Trans. **89**–**D**(2), 700–708 (2006)
12. Flocchini, P., Pagli, L., Prencipe, G., Santoro, N., Widmayer, P.: Computing all the best swap edges distributively. J. Parallel Distrib. Comput. **68**(7), 976–983 (2008)
13. Gualà, L., Proietti, G.: Exact and approximate truthful mechanisms for the shortest paths tree problem. Algorithmica **49**(3), 171–191 (2007)
14. Italiano, G.F., Ramaswami, R.: Maintaining spanning trees of small diameter. Algorithmica **22**(3), 275–304 (1998)
15. Ito, H., Iwama, K., Okabe, Y., Yoshihiro, T.: Single backup table schemes for shortest-path routing. Theor. Comput. Sci. **333**(3), 347–353 (2005)
16. Jordan, C.: Sur les assemblages de lignes. J. Reine Angew. Math **70**(185), 81 (1869)
17. Nardelli, E., Proietti, G., Widmayer, P.: A faster computation of the most vital edge of a shortest path. Inf. Process. Lett. **79**(2), 81–85 (2001)
18. Nardelli, E., Proietti, G., Widmayer, P.: Swapping a failing edge of a single source shortest paths tree is good and fast. Algorithmica **35**(1), 56–74 (2003)
19. Pettie, S.: Sensitivity analysis of minimum spanning trees in sub-inverse-ackermann time. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 964–973. Springer, Heidelberg (2005). https://doi.org/10.1007/11602613_96
20. Tarjan, R.E.: Sensitivity analysis of minimum spanning trees and shortest path trees. Inf. Process. Lett. **14**(1), 30–33 (1982)
21. Wu, B.Y., Hsiao, C.Y., Chao, K.M.: The swap edges of a multiple-sources routing tree. Algorithmica **50**(3), 299–311 (2008)

# Dynamic Networks

# A Generic Framework for Computing Parameters of Sequence-Based Dynamic Graphs

Arnaud Casteigts[1(✉)], Ralf Klasing[1(✉)], Yessin M. Neggaz[2(✉)], and Joseph G. Peters[3(✉)]

[1] LaBRI, CNRS, University of Bordeaux, Talence, France
{arnaud.casteigts,ralf.klasing}@labri.fr
[2] IRIT - SMAC Team, University of Toulouse, Toulouse, France
yessin.neggaz@gmail.com
[3] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
peters@cs.sfu.ca

**Abstract.** We presented in [12] an algorithm for computing a parameter called $T$*-interval connectivity* of dynamic graphs which are given as a sequence of static graphs. This algorithm operates at a high level, manipulating the graphs in the sequence as atomic elements with two types of operations: a *composition* operation and a *test* operation. The algorithm is optimal in the sense that it uses only $O(\delta)$ composition and test operations, where $\delta$ is the length of the sequence. In this paper, we generalize this framework to use various composition and test operations, which allows us to compute other parameters using the same high-level strategy that we used for $T$-interval connectivity. We illustrate the framework through the study of three minimization problems which refer to various properties of dynamic graphs, namely Bounded-Realization-of-the-Footprint, Temporal-Connectivity, and Round-Trip-Temporal-Diameter.

**Keywords:** Dynamic networks · Property testing · Generic algorithms Temporal connectivity

## 1 Introduction

Dynamic networks consist of entities making contact over time with one another. The types of dynamics resulting from these interactions are varied in scale and nature. For instance, some of these networks remain connected at all times [21]; others are always disconnected [18] but still offer some kind of connectivity over

Part of this work was done while Joseph Peters was visiting the LaBRI as a guest professor of the University of Bordeaux. This work was partially funded by the ANR projects DISPLEXITY (ANR-11-BS02-014) and ESTATE (ANR-16-CE25-0009-03). This study has been carried out in the frame of "The Investments for the Future" Programme IdEx Bordeaux CPU (ANR-10-IDEX-03-02). The work of Joseph Peters was partially supported by NSERC of Canada.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 321–338, 2017.
https://doi.org/10.1007/978-3-319-72050-0_19

time and space (*temporal* connectivity); others are recurrently connected, periodic, etc. All of these contexts can be represented as properties of dynamic graphs (also called time-varying graphs, evolving graphs, or temporal graphs). A number of such classes were identified in recent literature and organized into a hierarchy in [11]. Each of these classes corresponds to specific properties which play a role either in the complexity or in the feasibility of distributed problems. For example, it was shown in [10] that if the edges are *recurrent* (i.e. if an edge appears once, then it will reappear infinitely often), denoted class $\mathcal{R}$, then such a property guarantees the feasibility of a certain type of optimal broadcast with termination detection (namely, *foremost* broadcast). However, it is not sufficient to satisfy other measures of optimality, such as *shortest* or *fastest* broadcast. Strengthening the assumption to having a *bound* on the reappearance time (class $\mathcal{B}$) makes it possible to achieve shortest broadcast, and the even stronger assumption of having *periodic* edges (class $\mathcal{P}$) enables fastest broadcast. These three classes have been shown to play a role in a variety of problems (see e.g. [1,15,22]). Another important class, which is less constrained (and thus more general) is the class of all graphs with recurrent temporal connectivity (i.e. all nodes can recurrently reach each other through journeys), corresponding to class $\mathcal{C}_5$ in the hierarchy of [11]. This property is very general, and it is used (implicitly or explicitly) in a number of recent studies addressing distributed problems in highly-dynamic environments [4–6,14]. Interestingly, this property was considered more than three decades ago by Awerbuch and Even [2].

Given a dynamic graph, a natural question to ask is to which of the classes this graph belongs, or what related property it satisfies. These questions are interesting in several respects. Firstly, most of the known classes correspond to necessary or sufficient conditions for given distributed problems or algorithms (broadcast, election, spanning trees, token forwarding, etc.). Thus, being able to classify a graph in the hierarchy is useful for determining which problems can be solved on that graph. Furthermore, it is useful for choosing a good algorithm in settings where some properties are guaranteed (as in the above example with classes $\mathcal{R}, \mathcal{B}$, and $\mathcal{P}$). Hence, when targeting a given scenario from the real world, an algorithm designer may first record some topological traces from the target environment and then test which useful properties are satisfied. A growing amount of research is now focusing on testing properties (or computing structures) in dynamic graphs. A seminal example is the computation of foremost, shortest, or fastest journeys [7], the algorithms of which can also be used to test membership in a number of dynamic graph classes [8]. More recent examples include computing reachability graphs [3,24], enumerating maximal cliques [23], and establishing the hardness of computing metrics like *temporal diameter* (that is, how long it takes in the worst case to communicate through journeys) when the evolution is not known in advance [17].

In a previous paper [12], we focused on a property called $T$*-interval connectivity* [20], which captures two aspects of a network, *stability* and *connectivity*, and was shown to play a role in several distributed problems, such as determining the size of a network or computing a function of the initial inputs of the nodes.

$T$-interval connectivity (Class $\mathcal{C}_{10}$ in [11]) generalizes the class of dynamic graphs that are connected at all time instants [21] (Class $\mathcal{C}_9$ in [11]). The definition of $T$-interval connectivity is closely related to a representation of a network as a sequence of graphs $\mathcal{G} = (G_1, G_2, ..., G_\delta)$ which correspond to the state of the topology at increasing time instants (also called *untimed* evolving graphs [7]). Informally, $T$-interval connectivity requires that, for every $T$ consecutive graphs in $\mathcal{G}$, there exists a common connected spanning subgraph. In [12], we proposed a high-level algorithm for finding the largest $T$ such that a given sequence $\mathcal{G}$ is $T$-interval connected. We also addressed the related decision problem of testing if $\mathcal{G}$ is $T$-interval connected for given $T$. The approach in [12] focuses on high-level strategies in which the graphs in the sequence are considered to be atomic elements and the algorithm only uses two high-level operations on these elements: the intersection of two graphs, and testing if a given graph is connected. We showed that both the maximization and decision versions of the problem can be solved using only a linear number (in the length $\delta$ of the sequence) of such operations. The technique is based on a walk in a hierarchy, the elements of which are graphs that represent the intersections of various subsequences of $\mathcal{G}$.

### 1.1 Contributions

In this paper, we show that the high-level logic of the algorithm from [12] is actually quite general and can be used to compute a number of parameters in addition to $T$-interval connectivity by replacing the *intersection* and *connectivity test* operations by other operations. We begin by abstracting the two operations into a *composition* operation, which defines the hierarchy of elements in which the walk is performed, and a *test* operation, which determines the choices made by the walk. We investigate both the maximization and minimization of graph parameters and illustrate our framework with four instantiations of the operations: one solves a maximization problem ($T$-interval connectivity) and three instantiations solve the following minimization problems concerning temporal properties of recognized importance.

First, we consider the class of dynamic graphs with *time-bounded reappearance of edges*. A graph has time-bounded edge reappearance with bound $b$ if the time between two appearances of the same edge in the graph $\mathcal{G}$ is at most $b$. This property, together with the knowledge of $n$ (the number of nodes) and $b$, allows the feasibility of shortest broadcast with termination detection [9]. We consider the problem Bounded-Realization-of-the-Footprint of finding the smallest bound $b$ such that $\mathcal{G}$ has time-bounded edge reappearance, *i.e.* the smallest $b$ such that every edge that appears in the sequence $\mathcal{G}$ appears at least once in every subsequence of length $b$ of $\mathcal{G}$.

Then, we look at the class of dynamic graphs with *temporal connectivity* where a journey (temporal path) exists from any node to all other nodes. In this class of graphs, any node can perform a broadcast to all other nodes and can collect information from all the other nodes. The concept of temporal connectivity is relatively old and dates back at least to the article [2]. We consider the minimization problem Temporal-Diameter of finding the *temporal diameter* of a

given dynamic graph $\mathcal{G}$, i.e. the smallest duration in which there exist journeys (temporal paths) from any node to all other nodes.

Finally, we are interested in the class of dynamic graphs with *round-trip temporal connectivity* meaning that a back-and-forth journey exists from any node to all other nodes. This class characterizes an important property of distributed solutions for information collection problems that require termination detection [11]. We investigate the problem Round-Trip-Temporal-Diameter of computing the *round trip diameter* of a given graph $\mathcal{G}$, i.e. the smallest duration in which there exist back-and-forth journeys from any node to all other nodes.

## 2 Definitions and Observations

Let $\mathcal{G}$ be a graph sequence $\{G_1, G_2, ..., G_\delta\}$ such that $G_i = (V, E_i)$ represents the network topology at time $i$. Note that $V$ does not vary; only the edges change. We assume that the changes between two consecutive graphs in the sequence are arbitrary. Let $P$ be a boolean predicate (hereafter called *property*) defined on a consecutive subsequence $\{G_i, G_{i+1}, ..., G_j\} \subseteq \mathcal{G}$.

**Definition 1.** *The* minimization problem on $\mathcal{G}$ with respect to $P$ *is the problem of finding the smallest* $k$ *such that* $\forall i \in [1, \delta - k + 1]$, $\{G_i, G_{i+1}, ..., G_{i+k-1}\}$ *has property* $P$ *(in other words, any subsequence of* $\mathcal{G}$ *of length* $k$ *satisfies* $P$*).*

**Definition 2.** *The* maximization problem on $\mathcal{G}$ with respect to $P$ *is the problem of finding the largest* $k$ *such that* $\forall i \in [1, \delta - k + 1]$, $\{G_i, G_{i+1}, ..., G_{i+k-1}\}$ *has property* $P$*.*

We present here a general strategy for minimization and maximization problems that relies on a *composition hierarchy* of elements which is computed on demand using a *composition operation.*

**Definition 3 (Composition hierarchy and test operation).** *An* element $G_{(i,j)} : i \leq j$ *of a composition hierarchy is a graph from which one can determine whether the sequence* $\{G_i, G_{i+1}, ..., G_j\}$ *satisfies a predicate* $P$ *using a* test operation *which maps any element into* $\{true, false\}$*:* $test(G_{(i,j)}) = true$ *iff the sequence* $\{G_i, G_{i+1}, ..., G_j\}$ *satisfies* $P$*. The initial* $G_i$*'s are not elements of the hierarchy themselves, but all* $G_{(i,i)}$*'s are.*

*A hierarchy of elements consists of rows denoted* $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^\delta$ *where* $\mathcal{G}^k = \{G_{(1,k)}, G_{(2,k+1)}, \ldots, G_{(\delta-k+1,\delta)}\}$*. We use* $\mathcal{G}^k[i]$ *to denote the* $i^{th}$ *element of row* $\mathcal{G}^k$*, that is the element* $G_{(i,i+k-1)}$*. The first row* $\mathcal{G}^1$ *of the hierarchy corresponds to the graphs of the sequence* $\mathcal{G}$ *(or to simple transformations of these graphs); that is,* $G_{(i,i)}$ *corresponds to* $G_i$*. An example of a hierarchy in which elements are intersection graphs is shown in Fig. 1.*

**Definition 4 (Composition operation).** *A* composition operation $\circ$ *is a binary operation that maps two elements of the hierarchy into another element:* $G_{(i,j)} \circ G_{(i',j')} = S$ *where* $S$ *is the element that relates to the sequence* $\{G_i, G_{i+1}, ..., G_j, G_{i'}, G_{i'+1}, ..., G_{j'}\}$.

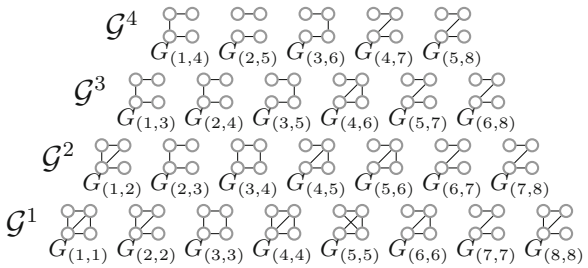



**Fig. 1.** Example of partial hierarchy with intersection graphs as elements.

**Observation 1.** *A minimization (resp. maximization) problem amounts to finding the lowest (highest) row $\mathcal{G}^k$ in which all elements $\{G_{(1,k)}, G_{(2,k+1)}, \ldots, G_{(\delta-k+1,\delta)}\}$ satisfy the test.*

The general framework that we propose makes it possible to solve minimization or maximization problems by focusing only on the composition and test operations, while the high-level logic of the algorithm remains the same. More precisely, there is one high-level algorithm for minimization problems, and another for maximization problems.

**Observation 2 (Requirements).** *For a minimization or a maximization problem relative to some property $P$ to be solvable within our framework, the following conditions must hold on the composition operation $\circ$ and the test operation* `test`*:*

*(1)* $test(G_{(i,j)}) = true \Leftrightarrow \{G_i, G_{i+1}, \ldots, G_{j-1}, G_j\}$ *satisfies* $P$*;*
*(2) The composition operation $\circ$ is associative, that is* $(G_{(i,j)} \circ G_{(i',j')}) \circ G_{(i'',j'')} = G_{(i,j)} \circ (G_{(i',j')} \circ G_{(i'',j'')})$.

*Only for maximization problems:*

*(3')* *If* $test(G_{(i,j)}) = true$ *then* $test(G_{(i',j')}) = true$ *for all* $i' \geq i$ *and* $j' \leq j$.

*Only for minimization problems:*

*(3")* *If* $test(G_{(i,j)}) = true$ *then* $test(G_{(i',j')}) = true$ *for all* $i' \leq i$ *and* $j' \geq j$.

## 3 Generic Algorithm

We propose a strategy based on the generic composition and test operations defined above. The algorithm is then instantiated in Sect. 4 to solve three specific minimization problems and one maximization problem by plugging in the appropriate operations. The strategy relies on the concept of *ladder*. Informally, a ladder is a sequence of elements that "climbs" the composition hierarchy bottom-up.

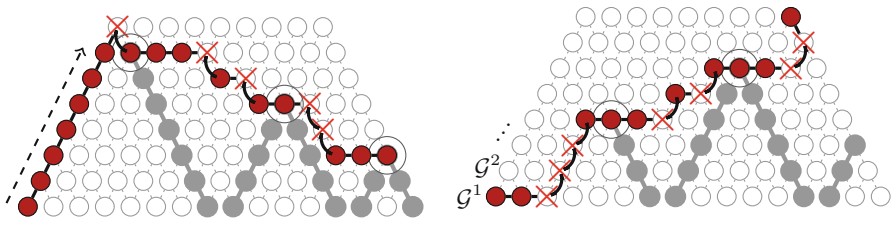


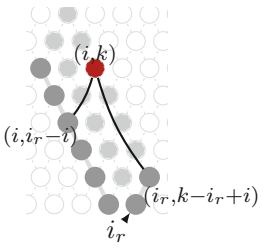

**Fig. 2.** Example of execution of the algorithm in (a) the maximization case (b) the minimization case.

**Definition 5.** *The* right ladder of length $l$ at index $i$, *denoted by* $\mathcal{R}^l[i]$, *is the sequence of elements* $\{\mathcal{G}^k[i],\ k = 1, 2, \ldots, l\}$. *The* left ladder of length $l$ at index $i$, *denoted by* $\mathcal{L}^l[i]$, *is the sequence* $\{\mathcal{G}^k[i-k+1],\ k = 1, 2, \ldots, l\}$.

**Lemma 1** [12]. *A ladder of length $l$ can be computed using $l-1$ binary compositions by computing each element as the composition of the preceding element in the ladder and an element in $\mathcal{G}^1$.*

**Lemma 2** [12]. *Given a left ladder of length $l_\ell$ at index $i_\ell$ and a right ladder of length $l_r$ at index $i_r = i_\ell + 1$. For any pair $(i,k)$ such that $i_r - l_\ell \leq i < i_r$ and $i_r - i < k \leq i_r - i + l_r$, $\mathcal{G}^k[i]$ can be computed by a single composition operation, namely $\mathcal{G}^k[i] = \mathcal{G}^{i_r - i}[i] \circ \mathcal{G}^{k - i_r + i}[i_r]$.*

Informally, the constraints $i_r - l_\ell \leq i < i_r$ and $i_r - i < k \leq i_r - i + l_r$ in Lemma 2 define a rectangle delimited by two ladders and two lines that are parallel to the two ladders as shown in the figure to the right. The pairs $(i,k)$ defined by the constraints, shown in light grey in the figure, include all pairs that are strictly inside the rectangle, and all pairs on the parallel lines, but pairs on the two ladders are excluded.




### 3.1 Informal Description of the Algorithm

We describe the algorithm with reference to Fig. 2a and b that respectively show examples of executions in the maximization case and the minimization case (see Algorithm 1 for details).

The algorithm takes as input a boolean problem type `problem` $\in \{min, max\}$, a dynamic graph $\mathcal{G}$, a composition operation $\circ$, and a test operation `test`. It starts by computing the first element $\mathcal{G}^1[1]$ and then traverses the hierarchy from left to right by computing a new adjacent element at each step: the next element in the same row, or the element with the same index in the row above, or the next element in the row below, depending on `problem` and the result of the `test` operation on the current element. We will call this traversal process a *walk*.

In the maximization case, the walk starts at the element $\mathcal{G}^1[1]$ and builds a right ladder incrementally until the test is negative (first loop, lines 3 ff. of Algorithm 1). If $\mathcal{G}^\delta[1]$ is reached and $test(\mathcal{G}^\delta[1]) = true$, then the execution terminates returning $\delta$. Otherwise, suppose that $test(\mathcal{G}^{k+1}[1]) = false$ for some $k$. Then $k$ is an upper bound on the maximization parameter of $\mathcal{G}$ and the walk drops down a level to $\mathcal{G}^k[2]$ which is the next element in row $k$ that needs to be tested. The walk proceeds rightward on row $k$ by computing at each step a new element in the row while the test is true. However, every time the test is negative, the walk drops down by one row. If the walk eventually reaches the rightmost element $\mathcal{G}^k[\delta - k + 1]$ of some row $k$ and $test(\mathcal{G}^k[\delta - k + 1]) = true$, then the algorithm terminates returning $k$. Otherwise the walk will terminate at an element $\mathcal{G}^1[i]$ that does not satisfy the test. In this case, the algorithm returns 0 indicating that the dynamic graph $\mathcal{G}$ does not have the property.

In the minimization case, the walk goes up in the composition hierarchy if the test is negative, otherwise it moves forward in the same row. If the walk hits the right side of the hierarchy and the last visited element $\mathcal{G}^k[\delta-k+1]$ in the row $\mathcal{G}^k$ satisfies the `test` operation, then it terminates and returns $k$. Otherwise, it terminates returning $k+1$ (Observation 2, requirement (3")). If the walk reaches $\mathcal{G}^1[\delta]$ and the test is negative, then the algorithm outputs 0 indicating that the dynamic graph $\mathcal{G}$ does not have the property.

**Computing elements of the hierarchy (function** *compute*()**).** The elements resulting from the walk (red/dark elements) are computed based on ladders (intermediate elements, in grey in Fig. 2a and b) as follows. When the walk moves one step forward in the same row, the next element is computed from a right ladder and a left ladder (*e.g.* $\mathcal{G}^4[6] = \mathcal{G}^2[6] \circ \mathcal{G}^2[8]$ in Fig. 2b) or from the ladder to which it belongs and an adjacent bottom element (*e.g.* $\mathcal{G}^5[9] = \mathcal{G}^1[9] \circ \mathcal{G}^4[10]$ in Fig. 2b). Intermediate elements *i.e. ladders* (in grey in Fig. 2a and b) are computed, according to Lemma 1, by incrementally composing an element $G_{(i,j)}$ with the adjacent bottom element $G_{(i-1,i-1)}$ (left ladder) or $G_{(j+1,j+1)}$ (right ladder), providing useful shortcuts in the construction. Suppose that $\mathcal{G}^k[i]$ is the first element to be computed where no element $\mathcal{G}^{k'}[i]$ with $k' < k$ has been computed. The first ladder built is $\mathcal{L}^k[k + i - 1]$ of length $k$ ending at $\mathcal{G}^k[i]$ ($\mathcal{G}^7[2]$ in Fig. 2a, $\mathcal{G}^4[4]$ in Fig. 2b). Differently from left ladders, right ladders are constructed gradually as the walk proceeds. Each time that the walk moves right to a new index, the current right ladder is incremented (a new element is added to the ladder) and the new top element of this right ladder is used immediately to compute the element at the current index in the walk (using Lemma 2). This continues until the walk crosses the current right ladder, on an element $\mathcal{G}^k[i]$ ($\mathcal{G}^6[8]$ in Fig. 2b), at which time a left ladder $\mathcal{L}^k[k + i - 1]$ is built to compute $\mathcal{G}^k[i]$ and to be used to compute the next elements on the walk.

This generic algorithm has the following property which is crucial for the correctness of two of the problems described in Sect. 4.

```
   Input: problem ∈ {min, max}, 𝒢, composition operation ∘, test operation test
 1 i ← 1                                              // current index in the row
 2 k ← 1                                              // current row

 3 if problem = max then
 4 |  compute(𝒢^k[i])
 5 |  while test(𝒢^k[i]) do
 6 |  |  if k = δ then
 7 |  |  |  return k
 8 |  |  else
 9 |  |  |  k++; compute(𝒢^k[i])
10 |  |  k−−; i++
11 while 1 ≤ k ≤ δ do
12 |  compute(𝒢^k[i])
13 |  if test(𝒢^k[i]) then
14 |  |  if i = δ − k + 1 then
15 |  |  |  return k
16 |  |  else
17 |  |  |  i++
18 |  else
19 |  |  switch problem do
20 |  |  |  case max do
21 |  |  |  |  k−−; i++
22 |  |  |  case min do
23 |  |  |  |  if i = δ − k + 1 then
24 |  |  |  |  |  return k + 1
25 |  |  |  |  else
26 |  |  |  |  |  k++
27 return 0
```

**Algorithm 1.** Generic algorithm for minimization and maximization problems

**Lemma 3 (Disjoint sequences property).** *If the algorithm performs a composition of two elements $G_{(i,j)}$ and $G_{(i',j')}$, then the corresponding sequences $\{G_i, G_{i+1}, \ldots, G_j\}$ and $\{G'_i, G_{i'+1}, \ldots, G_{j'}\}$ are disjoint and consecutive. That is, in any execution, $G_{(i,j')} = G_{(i,j)} \circ G_{(i',j')} \Rightarrow j = i' - 1$.*

*Proof.* According to the algorithm, each element of the composition hierarchy is computed from: (1) two elements of two different ladders, a left one and a right one, or (2) an element of a ladder and an element in the first row. In both cases the two sequences covered by the two elements used in the computation are disjoint and consecutive, so in any execution, $G_{(i,j')} = G_{(i,j)} \circ G_{(i',j')} \Rightarrow j = i' - 1$. □

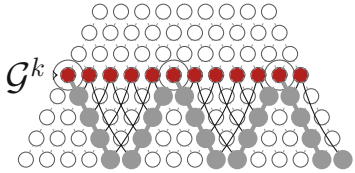



**Fig. 3.** Example of the execution of the algorithm for the decision variant.

**Lemma 4.** *Let $\mathcal{G}^k[\delta - k + 1]$ be the last visited element at the termination of the algorithm. If $test(\mathcal{G}^k[\delta - k + 1]) = true$ , then $\forall i \in [1, \delta - k]$, $test(G^k[i]) = true$ (all elements in the row $\mathcal{G}^k$).*

*Proof.* According to the algorithm, for any element $G_{(i,j)}$ above (below) the walk in the minimization (maximization) case, there exists a computed element $G_{(i',j')}$ in the walk such that $i' \leq i \wedge j' \geq j$ ($i' \geq i \wedge j' \leq j$) and $test(G_{(i',j')}) = true$. According to Observation 2 (requirements (3') and (3")), any element above (below) the walk in the minimization (maximization) case satisfies the test operation. □

**Decision variant.** The algorithm for the decision variant of each problem (*i.e.* for a given $k$, answer *true* if any sequence of length $k$ in the dynamic graph $\mathcal{G}$ has the property $P$, answer $false$ otherwise) can be deduced readily from the algorithm for the minimization/maximization variant. The algorithm gradually computes elements of row $k$ from left to right, starting at $\mathcal{G}^k[1]$, as shown in Fig. 3. If an element that does not satisfy the test operation is found, the algorithm returns $false$ and terminates. If the algorithm reaches the last element in the row, i.e. $\mathcal{G}^k[\delta - k + 1]$, and it satisfies the test operation, then it returns *true*. The elements $\mathcal{G}^k[1], \mathcal{G}^k[2], \ldots, \mathcal{G}^k[\delta - k + 1]$ are computed based on ladders.

**Theorem 1.** *The generic algorithm has a cost of O(δ) composition and test operations.*

*Proof.* The ranges of the indices covered by the left ladders that are constructed by the process are disjoint, so their total length is $O(\delta)$. With the computation of each new element in a right ladder, the walk moves closer to the right side of the hierarchy, so the total length of the right ladders is also $O(\delta)$. According to Lemma 2, any element can be computed using a single composition operation based on ladders. According to the algorithm, the number of elements computed by the walk is $O(\delta)$ and any computed element is tested at most once. This establishes that this algorithm has a cost of $O(\delta)$ composition operations and test operations. □

**Online version.** The generic algorithm can be adapted to an online setting in which the sequence of graphs $G_1, G_2, G_3, \ldots$ of a dynamic graph $\mathcal{G}$ is processed in the order that the graphs are received. For the decision problem, the algorithm

cannot provide an answer until at least $k$ graphs have been received. When the $k^{th}$ graph is received, the algorithm builds the first left ladder using $k-1$ compositions. It can then perform a test and answer whether or not the sequence has the property so far. After this initial period, a test can be performed for the $k$ most recently received graphs (by performing a test on the corresponding element in row $T$) after the receipt of each new graph. The same logic is followed for minimization and maximization problems.

**Theorem 2.** *The online generic algorithm has an amortized cost of O(1) composition and test operations per graph received.*

*Proof.* At no time during the execution of the algorithm does the number of compositions performed to build left ladders exceed the number of graphs received and the same is true for right ladders. The number of elements on the walk that are not on ladders never exceeds the number of graphs received, and each can be computed with one composition by Lemma 2. Only elements on the walk are tested. In summary, the amortized cost is O(1) composition and test operations for each graph received. □

## 4 Illustration of the Framework

We illustrate the general framework by solving one maximization problem: Interval-Connectivity and three minimization problems: Bounded-Realization-of-the-Footprint, Temporal-Diameter, and Round-Trip-Temporal-Diameter. We define each problem within the framework and provide the corresponding operations for composition and test.

### 4.1 $T$-Interval Connectivity (Maximization)

A dynamic graph $\mathcal{G}$ is said to be *$T$-interval connected* if for any $t \in [1, \delta - T + 1]$ all graphs in $\{G_t, G_{t+1}, ..., G_{t+T-1}\}$ share a common connected spanning subgraph. We consider the problem Interval-Connectivity of finding the smallest duration $T$ for which the dynamic graph $\mathcal{G}$ is $T$-interval connected.

**Composition and test operations.** By using the *intersection* of two elements as the composition operation (starting with $\{G_{(i,i)}\} = \{G_i\}$), a hierarchy of intersection graphs (Fig. 1) as elements can be used to solve Interval-Connectivity which is the problem of finding the highest row $\mathcal{G}^T$ in which every element $\mathcal{G}^T[i]$, $i \in [1, \delta - T + 1]$, is connected. So, the composition operation is *intersection* and the test operation is *connectivity test.*

**Observation 3 (Cost of the operations).** *Using an adjacency list data structure, the binary intersection of two elements $G_{(i,j)}$ and $G_{(i',j')}$ can be computed in time $O(\min(|E(G_{(i,j)})|, |E(G_{(i',j')})|))$. Testing connectivity of an undirected graph can also be done in time $O(|E(G_{(i,j)})|)$ by building a depth-first search tree from an arbitrary node to test whether all nodes are reachable.*

### 4.2 Bounded Realization of the Footprint (Minimization)

The *footprint* $G$ of a dynamic graph $\mathcal{G}$ is the graph that contains all the edges that appear at least once, that is $\cup\{G_1, G_2, ..., G_\delta\}$. We consider the problem of finding the smallest duration $b$ such that in any window of length $b$, all edges of $G$ appear at least once (Bounded-Realization-of-the-Footprint). The problem then amounts to finding the lowest row $\mathcal{G}^b$ in which every element $\mathcal{G}^b[i]$, $i \in [1, \delta - b + 1]$, equals the footprint $G$.

**Composition and test operations.** Finding these operations is straightforward. By taking the *union* of two elements as the composition operation (starting with $G_{(i,i)} = G_i$), it follows that the lowest row $\mathcal{G}^b$ such that all elements *equal* the footprint indicates, by definition, that the answer is $b$. So, the composition operation is *union* and the test operation is *equality to footprint*.

**Observation 4 (Cost of the operations).** *Using an adjacency matrix representation, the union operation and the equality test can be performed in* $O(|V|^2)$ *time.*

### 4.3 Temporal Diameter (Minimization)

A dynamic graph might never be connected at one time, and yet offer a form of connectivity over time based on journeys (temporal paths). Informally, a journey is a path whose edges are crossed at non-decreasing (or increasing) times, with possible pauses at intermediate nodes. The edges need not be all present simultaneously. If at most one edge can be crossed at a time (*i.e.* the crossing times are strictly increasing), then we refer to the journey as being *strict*. Formally, journeys can be defined in various ways, depending on the graph formalism used. In sequence-based models like evolving graphs, it is defined as follows.

**Definition 6 (Journey).** *A* journey $\mathcal{J}$ *from* $u$ *to* $v$ *in* $\mathcal{G}$ *is a sequence of edges* $e_1, e_2, \ldots, e_p$ *connecting* $u$ *to* $v$ *through intermediate vertices and a corresponding sequence of non-decreasing indices* $t_1, t_2, \ldots, t_p$ *such that* $e_i \in E(G_{t_i})$. *In a* strict journey, *the sequence* $t_1, t_2, \ldots, t_p$ *is strictly increasing. The existence of a journey from* $u$ *to* $v$ *is denoted* $u \rightsquigarrow v$. *We note* $\text{departure}(\mathcal{J}) = t_1$ *and* $\text{arrival}(\mathcal{J}) = t_p$.

The distinction between strict and non-strict journeys actually boils down to deciding if the latency of communication is neglected or not. In either case, one can define the concept of *temporal diameter* (at time $t$) as the smallest $d$ such that for all nodes $u$ and $v$, there exists a journey from $u$ to $v$ in the sequence $\{G_t, G_{t+1}, ..., G_{t+d-1}\}$. We consider here the problem Temporal-Diameter of finding the smallest $d$ such that the *temporal diameter* of $\mathcal{G}$ is less than or equal to $d$ at *every* time $t \leq \delta - d + 1$. In other words, any subsequence of $\mathcal{G}$ of length $d$ is temporally connected. Several solutions exist for this and similar problems (see e.g. [24]), which operate at a lower level of abstraction. Here, we show how the problem fits elegantly within our proposed framework. More specifically, we consider the case of *non-strict* journeys, which is slightly more difficult and contains as a subproblem the case of strict journeys.

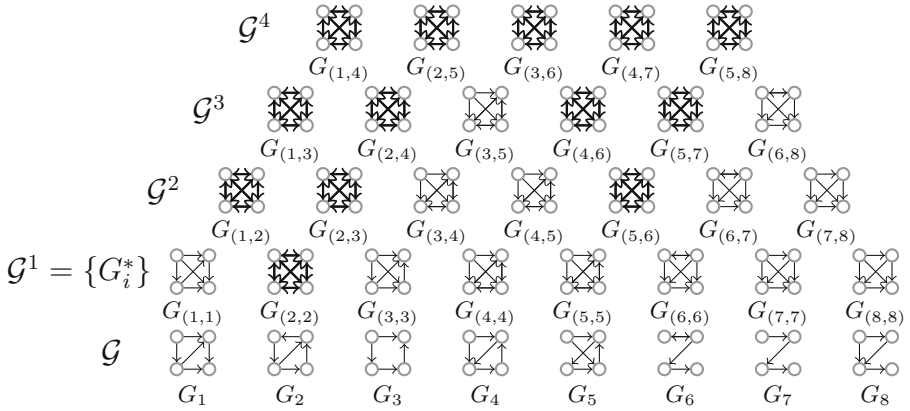



**Fig. 4.** Example of a transitive closure hierarchy for a given dynamic graph $\mathcal{G}$ of length $\delta = 8$.

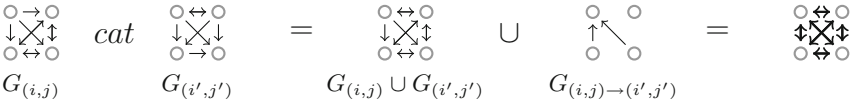



**Fig. 5.** Example of concatenation of transitive closures. *Edges in* $G_(i,j) \to (i',j')$ are added after the union.

**Definition 7 (Transitive closure).** *The* transitive closure *of the dynamic graph* $\mathcal{G}$ *is the static directed graph* $\mathcal{G}^* = (V, E^*)$ *such that* $(u,v) \in E^* \Leftrightarrow u \rightsquigarrow v$.

The composition hierarchy built here is one of *transitive closures* of journeys. Figure 4 shows an example. For this problem, each bottom element $G_{(i,i)}$ is not equal to $G_i$; instead, it corresponds to the "classical" *transitive closure* of $G_i$, i.e. the graph $G_i^*$ built on the same vertex set as $G_i$, such that an *edge* exists between $u$ and $v$ in $G_i^*$ if and only if a *path* exists between $u$ and $v$ in $G_i$ (the $G_{(i,i)}$'s are computed gradually as the algorithm progresses). Then, the answer is the smallest $d$ such that every element in row $\mathcal{G}^d$ of the hierarchy is a complete graph (i.e. every subsequence of $\mathcal{G}$ of length $d$ is temporally connected).

**Composition and test operations.** The composition hierarchy is built using *concatenation* of transitive closures, $cat(G_{(i,j)}, G_{(i',j')})$, with the restriction that $i' = j + 1$ (Lemma 3), defined as follows. First compute the union of both elements, then add an *additional edge* $(u,v)$ if there exists a node $w$ such that $(u,w) \in E(G_{(i,j)})$ and $(w,v) \in E(G_{(i',j')})$. See Fig. 5 for an example. Then, the *test* operation consists of determining if an element of the hierarchy (transitive closure) is a complete graph.

**Observation 5 (Cost of the operations).** *The union of two transitive closures* $G_{(i,j)}$ *and* $G_{(i',j')}$ *can be computed in time* $O(\max(|E(G_{(i,j)})|, |E(G_{(i',j')})|))$

*using an adjacency list data structure. The cost of the concatenation operation is dominated by the computation of the additional edges which costs* $O(|E(G_{(i',j')})| \cdot |V|)$. *The completeness test of a transitive closure* $G_{(i,j)}$ *can be done in constant time by checking* $|E(G_{(i,j)})|$ *which is maintained during the construction of the transitive closure graph.*

### 4.4 Round-Trip Temporal Diameter (Minimization)

We address here the more complex property of *round-trip temporal connectivity* defined by the existence of a back-and-forth journey from any node to all other nodes. The *round-trip temporal diameter* of a graph $\mathcal{G}$ at time $t$ is the smallest $d$ such that, in the sequence $\{G_t, G_{t+1}, ..., G_{t+d-1}\}$, there is a journey $\mathcal{J}(u, v)$ from any node $u$ in the graph to any other node $v$ *and* a journey $\mathcal{J}'(v, u)$ from $v$ to $u$ which starts after the arrival of the journey $\mathcal{J}(u, v)$. This does not mean that there is simply a succession of two temporally connected sequences. A back-and-forth journey from a node $u$ to a node $v$ can finish before a back-and-forth journey from a node $u'$ to a node $v'$ starts. Also, the time intervals of the two back-and-forth journeys can overlap. We consider the problem Round-Trip-Temporal-Diameter of finding the smallest $d$ such that the *round-trip temporal diameter* of $\mathcal{G}$ is less than or equal to $d$ at any time $t \leq \delta - d + 1$. For this problem, we consider the case of non-strict journeys.

**Definition 8 (Round trip transitive closure).** *A* round trip transitive closure $G_{(i,j)}$ *is the directed graph where* $(u, v) \in G_{(i,j)}$ *iff at least one journey* $u \rightsquigarrow v$ *exists in the sequence* $\{G_i, G_{i+1}, ..., G_j\}$. *The edges* $\{(u, v) \in E(G_{(i,j)})\}$ *are labelled with two times:* $\text{arrival}(u, v, G_{(i,j)})$ *is the earliest arrival of any journey in the sequence and* $\text{departure}(u, v, G_{(i,j)})$ *is the latest departure of any journey in the sequence. Labels on the same edge may or may not be the departure and arrival times of the same journey. Note that* $departure(u, v, G_{(i,i)}) = i$ *and* $arrival(u, v, G_{(i,i)}) = i$.

The composition hierarchy built for this problem is one of *round trip transitive closures* of journeys. Figure 6 shows an example of a round trip transitive closure hierarchy of a dynamic graph $\mathcal{G}$ of length $\delta = 3$. Labels *arr* and *dep* on an edge $u \xrightarrow{arr,dep} v$ (label on the destination/head end) represent respectively $arrival(u, v, G_{(i,j)})$ and $departure(u, v, G_{(i,j)})$. As for Temporal-Diameter, each bottom element $G_{(i,i)}$ corresponds to the "classical" *transitive closure* of $G_i$, i.e. the graph $G_i^*$ built on the same vertex set as $G_i$, such that an *edge* exists between $u$ and $v$ in $G_i^*$ if and only if a *path* exists between $u$ and $v$ in $G_i$. The labels of the edges are initialized with "$i, i$", which corresponds to the arrival and departure times of the corresponding journey(s). Then, the answer is the smallest $d$ such that every element in row $\mathcal{G}^d$ is a complete graph (i.e. every subsequence of $\mathcal{G}$ of length $d$ is round-trip temporally connected).

**Composition operation.** The composition operation in this case is the *concatenation of round trip transitive closures* $rtcat(G_{(i,j)}, G_{(i',j')})$ with the restriction

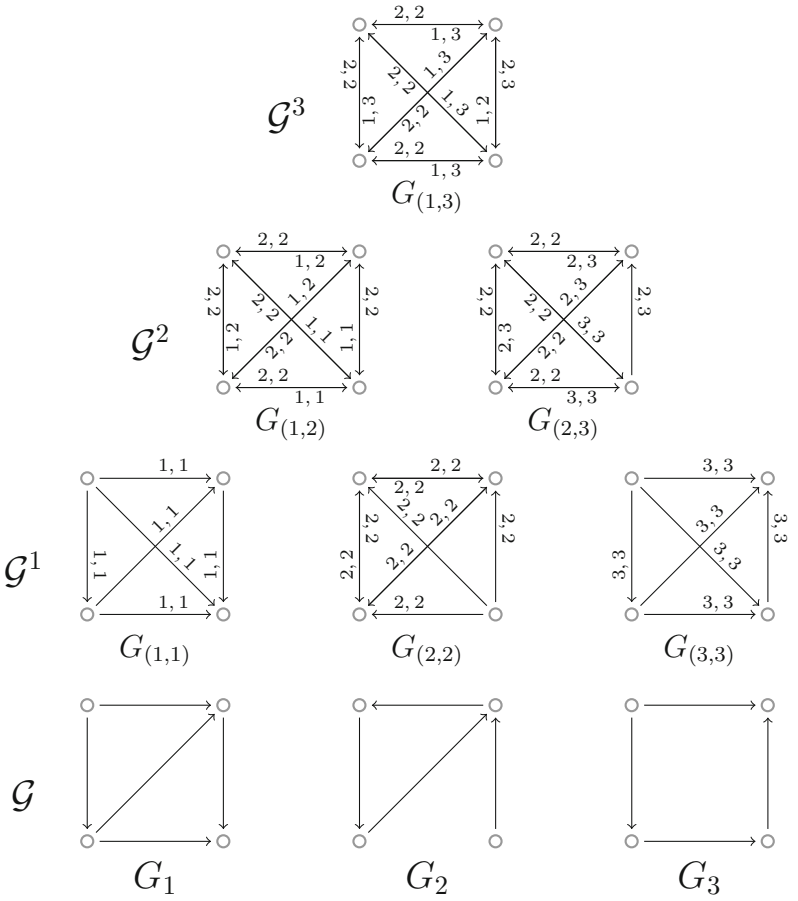



**Fig. 6.** Example of a round trip transitive closure hierarchy of a dynamic graph $\mathcal{G}$ of length $\delta = 3$. (*Arrival and departure times are on the head ends of the arrows.*)

that $i' = j + 1$ (Lemma 3). A composition is computed as follows. First, compute the graph $G^{\cup\circlearrowleft} = G_{(i,j)} \cup^{\circlearrowleft} G_{(i',j')}$ which is the union graph $G_{(i,j)} \cup G_{(i',j')}$ with $arrival(u, v, G^{\cup\circlearrowleft}) = \min(arrival(u, v, G_{(i,j)}), arrival(u, v, G_{(i',j')}))$ and $departure(u, v, G^{\cup\circlearrowleft}) = \max(departure(u, v, G_{(i,j)}), departure(u, v, G_{(i',j')})$ if $(u, v) \in G_{(i,j)} \cap G_{(i',j')}$. Otherwise, the edge is added with the initial arrival and departure times. A graph of *extra edges* $G_{(i,j)\to(i',j')}$ is then computed as follows: $(u, v) \in G_{(i,j)\to(i',j')}$ iff there exists a non-empty set of nodes $extra = \{w : (u, w) \in E(G_{(i,j)})$ and $(w, v) \in E(G_{(i',j')})\}$. The labels on an extra edge are $arrival(u, v, G_{(i,j)\to(i',j')}) = \min_{w\in extra}\{arrival(w, v, G_{(i',j')})\}$ and $departure(u, v, G_{(i,j)\to(i',j')}) = \max_{w\in extra}\{departure(u, w, G_{(i,j)})\}$. Finally, the round trip transitive closure $rtcat(G_{(i,j)}, G_{(i',j')}) = G^{\cup\circlearrowleft} \cup^{\circlearrowleft} G_{(i,j)\to(i',j')}$ (see Fig. 7).

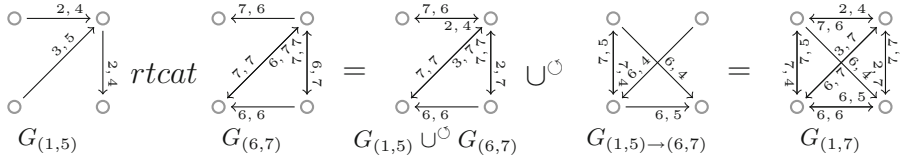



**Fig. 7.** Example of round trip transitive closures concatenation. (*Arrival and departure times are on the head ends of the arrows.*)

**Test operation.** The test operation used for this problem is the *round trip completeness test*, that is, test if the graph is complete and if $arrival(u, v, G_{(i,j)}) \leq departure(v, u, G_{(i,j)})$ for every edge $(u, v)$ in the graph.

**Observation 6 (Cost of the operations).** *As for the concatenation operation for* Temporal-Diameter*, the concatenation of two round trip transitive closures $G_{(i,j)}$ and $G_{(i',j')}$ can be computed in time $O(|E(G_{(i',j')})| \cdot |V|)$. The completeness test can be done in time $O(|E(G_{(i,j)})|)$ by verifying the condition on the times for each pair of edges $(u, v)$, $(v, u)$.*

### 4.5 Parallel Version

We define a subset of particular minimization and maximization problems that we call *symmetric problems* as follows.

**Definition 9 (Symmetric problems).** *A minimization or maximization problem is* symmetric *if it can be solved using a composition hierarchy of elements and a composition operation $\circ$ such that $G_{(i,j)} \circ G_{(i',j')} = G_{(i,j')}$ for all $1 \leq i \leq i' \leq j \leq j' \leq \delta$.*

Bounded-Realization-of-the-Footprint and $T$-Interval-Connectivity are examples of *symmetric problems*. We now present a strategy for *symmetric problems* that can be parallelized on a PRAM. We first describe the algorithms for a sequential machine (RAM). The general strategy is to compute only some of the rows of the composition hierarchy based on the following lemma. The proofs of Lemmas 5 and 6 are straightforward generaliztions of proofs in [12].

**Lemma 5.** *If some row $\mathcal{G}^k$ is already computed, then any row $\mathcal{G}^\ell$ for $k + 1 \leq \ell \leq 2k$ can be computed with $O(\delta)$ composition and test operations.*

**Decision variant.** Using Lemma 5, for a given $k$, we can incrementally compute rows $\mathcal{G}^{2^i}$ ("power rows") for all $i$ from 1 to $\lceil \log_2 k \rceil - 1$ without computing the intermediate rows. Then, we compute row $\mathcal{G}^k$ directly from row $\mathcal{G}^{2^{\lceil \log_2 k \rceil - 1}}$ (again using Lemma 5). This way, we compute $\lceil \log_2 k \rceil = O(\log \delta)$ rows using $O(\delta \log \delta)$ composition operations, after which we perform $O(\delta)$ tests.

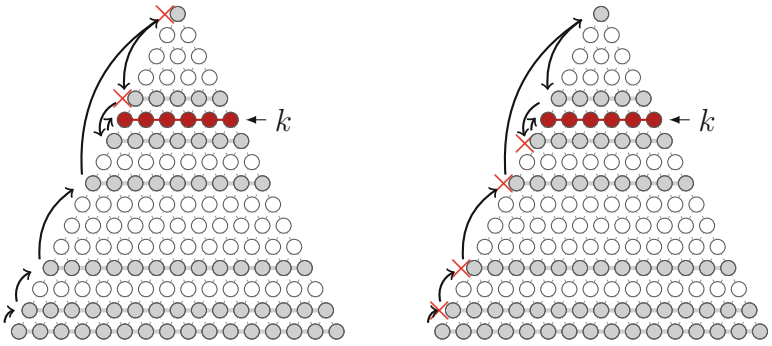



**Fig. 8.** Examples of the execution of the parallel version of the algorithm; *maximization case on the left and minimization case on the right.*

**Minimization and maximization variants.** For the maximization case, we incrementally compute rows $\mathcal{G}^{2^i}$ until we find a row that contains an element that does not satisfy the test operation (thus, a test is performed after each composition). By Lemma 5, each of these rows can be computed using $O(\delta)$ compositions. Suppose that row $\mathcal{G}^{2^{j+1}}$ is the first power row that contains an element that does not satisfy the test, and that $\mathcal{G}^{2^j}$ is the row computed before $\mathcal{G}^{2^{j+1}}$. Next, we do a binary search among the rows between $\mathcal{G}^{2^j}$ and $\mathcal{G}^{2^{j+1}}$ to find the highest row $\mathcal{G}^k$ such that all elements on this row satisfy the test. See Fig. 8 (left) for an illustration of the algorithm. The computation of each of these rows is based on row $\mathcal{G}^{2^j}$ and uses $O(\delta)$ compositions by Lemma 5. Overall, we compute at most $2\lceil \log_2 k \rceil = O(\log \delta)$ rows using $O(\delta \log \delta)$ compositions and the same number of tests.

For the minimization case, we follow the same principle. This time, we incrementally compute rows $\mathcal{G}^{2^i}$ while each row contains an element that does not satisfy the test. Suppose that row $\mathcal{G}^{2^{j+1}}$ is the first power row such that all elements on this row satisfy the test. Then, we do a binary search among the rows between $\mathcal{G}^{2^j}$ and $\mathcal{G}^{2^{j+1}}$ to find the lowest row $\mathcal{G}^k$ such that all elements on this row satisfy the test. See Fig. 8 (right) for an illustration of the algorithm.

**Lemma 6.** *If some row $\mathcal{G}^k$ is already computed, then any row between $\mathcal{G}^{k+1}$ and $\mathcal{G}^{2k}$ can be computed in $O(1)$ time on an EREW PRAM with $O(\delta)$ fprocessors.*

**Parallel version for the decision problems on an EREW PRAM.** The sequential algorithm for this problem computes $O(\log \delta)$ rows. By Lemma 6, each of these rows can be computed in $O(1)$ time on an EREW PRAM with $O(\delta)$ processors. Therefore, all of the rows (and hence all necessary compositions) can be computed in $O(\log \delta)$ time with $O(\delta)$ processors. The $O(\delta)$ tests for row $\mathcal{G}^k$ can be done in $O(1)$ time with $O(\delta)$ processors. Then, the processors can establish whether or not all elements in row $\mathcal{G}^k$ satisfy the test operation by computing the logical AND of the results of the $O(\delta)$ tests in time $O(\log \delta)$ on

a EREW PRAM with $O(\delta)$ processors using standard techniques (see [16,19]). The total time is $O(\log \delta)$ on an EREW PRAM with $O(\delta)$ processors.

**Parallel version for maximization and minimization problems on an EREW PRAM.** The sequential algorithm for this problem computes $O(\log \delta)$ rows. Differently from the decision version, a test is done for each of the computed elements (rather than just those of the last row) and it has to be determined for each computed row whether or not all of the elements satisfy the test. This takes $O(\log \delta)$ time for each of the $O(\log \delta)$ computed rows using the same techniques as for the decision version. The total time is $O(\log^2 \delta)$ on an EREW PRAM with $O(\delta)$ processors.

## 5 Conclusions

In this paper, we generalized the framework and the algorithm for Interval-Connectivity [12] to solve other problems on dynamic graphs. We studied the minimization problems of finding the temporal diameter and the round trip temporal diameter of a given dynamic graph $\mathcal{G} = \{G_1, G_2, ..., G_\delta\}$, and the problem of finding a bound on the footprint realization of $\mathcal{G}$. We proposed algorithms for these problems within the same framework.

In our study, we focused on algorithms using only two elementary operations, *composition* and *test* operations. This approach is suitable for a high-level study of these problems when the details of changes between successive graphs in a sequence are arbitrary. If the evolution of the dynamic graph is constrained in some ways (e.g., bounded number of changes between graphs), then one could benefit from the use of more sophisticated data structures to reduce the complexity of the algorithms.

A natural extension of our investigation would be a similar study for other classes and properties of dynamic networks, as identified in [11].

## References

1. Aaron, E., Krizanc, D., Meyerson, E.: DMVP: foremost waypoint coverage of time-varying graphs. In: Kratsch, D., Todinca, I. (eds.) WG 2014. LNCS, vol. 8747, pp. 29–41. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12340-0_3
2. Awerbuch, B., Even, S.: Efficient and reliable broadcast is achievable in an eventually connected network. In: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 278–281. ACM (1984)
3. Barjon, M., Casteigts, A., Chaumette, S., Johnen, C., Neggaz, Y.M.: Testing temporal connectivity in sparse dynamic graphs. CoRR abs/1404.7634 (2014). (A French version appeared in Proceedings of ALGOTEL 2014)
4. Bournat, M., Datta, A.K., Dubois, S.: Self-stabilizing robots in highly dynamic environments. In: Bonakdarpour, B., Petit, F. (eds.) SSS 2016. LNCS, vol. 10083, pp. 54–69. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49259-9_5

5. Bramas, Q., Tixeuil, S.: The complexity of data aggregation in static and dynamic wireless sensor networks. Inf. Comput. **255**, 369–383 (2017)
6. Braud-Santoni, N., Dubois, S., Kaaouachi, M.H., Petit, F.: The next 700 impossibility results in time-varying graphs. Int. J. Netw. Comput. **6**(1), 27–41 (2016)
7. Bui-Xuan, B., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. Int. J. of Found. Comput. Sci. **14**(2), 267–285 (2003)
8. Casteigts, A., Chaumette, S., Ferreira, A.: Characterizing topological assumptions of distributed algorithms in dynamic networks. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 126–140. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11476-2_11. Full version in CoRR, abs/1102.5529
9. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Measuring temporal lags in delay-tolerant networks. IEEE Trans. Comput. **63**(2), 397–410 (2014)
10. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Shortest, fastest, and foremost broadcast in dynamic networks. Int. J. Found. Comput. Sci. **26**(4), 499–522 (2015)
11. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. Int. J. Parallel, Emergent Distrib. Syst. **27**(5), 387–408 (2012)
12. Casteigts, A., Klasing, R., Neggaz, Y.M., Peters, J.G.: Efficiently testing $T$-interval connectivity in dynamic graphs. In: Paschos, V.T., Widmayer, P. (eds.) CIAC 2015. LNCS, vol. 9079, pp. 89–100. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18173-8_6
13. Casteigts, A., Klasing, R., Neggaz, Y.M., Peters, J.G.: Calcul de Paramètres Minimaux dans les Graphes Dynamiques. In: 19èmes Rencontres Francophones sur les Aspects Algorithmiques de Télécommunications (ALGOTEL) (2017)
14. Dubois, S., Kaaouachi, M.-H., Petit, F.: Enabling minimal dominating set in highly dynamic distributed systems. In: Pelc, A., Schwarzmann, A.A. (eds.) SSS 2015. LNCS, vol. 9212, pp. 51–66. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21741-3_4
15. Flocchini, P., Mans, B., Santoro, N.: On the exploration of time-varying networks. Theor. Comput. Sci. **469**, 53–68 (2013)
16. Gibbons, A., Rytter, W.: Efficient Parallel Algorithms. Cambridge University Press, Cambridge (1988)
17. Godard, E., Mazauric, D.: Computing the dynamic diameter of non-deterministic dynamic networks is hard. In: Gao, J., Efrat, A., Fekete, S.P., Zhang, Y. (eds.) ALGOSENSORS 2014. LNCS, vol. 8847, pp. 88–102. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46018-4_6
18. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: Proceedings of SIGCOMM, pp. 145–158 (2004)
19. JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley, Boston (1992)
20. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: Proceedings of STOC, pp. 513–522. ACM (2010)
21. O'Dell, R., Wattenhofer, R.: Information dissemination in highly dynamic graphs. In: Proceedings of DIALM-POMC, pp. 104–110. ACM (2005)
22. Raynal, M., Stainer, J., Cao, J., Wu, W.: A simple broadcast algorithm for recurrent dynamic systems. In: 2014 IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), pp. 933–939. IEEE (2014)
23. Viard, T., Latapy, M., Magnien, C.: Computing maximal cliques in link streams. Theor. Comput. Sci. **609**, 245–252 (2016)
24. Whitbeck, J., Dias de Amorim, M., Conan, V., Guillaume, J.L.: Temporal reachability graphs. In: Proceedings of MOBICOM, pp. 377–388. ACM (2012)

# Gathering in Dynamic Rings

Giuseppe Antonio Di Luna[1(✉)], Paola Flocchini[1], Linda Pagli[2], Giuseppe Prencipe[2], Nicola Santoro[3], and Giovanni Viglietta[1]

[1] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada
{gdiluna,flocchini,gvigliet}@uottawa.ca
[2] Dipartimento di Informatica, University of Pisa, Pisa, Italy
{linda.pagli,giuseppe.prencipe}@unipi.it
[3] School of Computer Science, Carleton University, Ottawa, Canada
santoro@scs.carleton.ca

**Abstract.** The *gathering* (or *multi-agent rendezvous*) problem requires a set of mobile agents, arbitrarily positioned at different nodes of a network to group within finite time at the same location, not fixed in advanced.

The extensive existing literature on this problem shares the same fundamental assumption: the topological structure does not change during the rendezvous or the gathering; this is true also for those investigations that consider faulty nodes. In other words, they only consider *static graphs*.

In this paper we start the investigation of gathering in *dynamic graphs*, that is networks where the topology changes continuously and at unpredictable locations.

We study the feasibility of gathering mobile agents, identical and without explicit communication capabilities, in a *dynamic ring* of anonymous nodes; the class of dynamics we consider is the classic *1-interval-connectivity*. We focus on the impact that factors such as *chirality* (i.e., a common sense of orientation) and *cross detection* (i.e., the ability to detect, when traversing an edge, whether some agent is traversing it in the other direction), have on the solvability of the problem; and we establish several results.

We provide a complete characterization of the classes of initial configurations from which the gathering problem is solvable in presence and in absence of cross detection and of chirality. The feasibility results of the characterization are all constructive: we provide distributed algorithms that allow the agents to gather within low polynomial time. In particular, the protocols for gathering with cross detection are time optimal.

We also show that cross detection is a powerful computational element. We prove that, without chirality, knowledge of the ring size is strictly more powerful than knowledge of the number of agents; on the other hand, with chirality, knowledge of n can be substituted by knowledge of k, yielding the same classes of feasible initial configurations.

From our investigation it follows that, for the gathering problem, the computational obstacles created by the dynamic nature of the ring can be overcome by the presence of chirality or of cross-detection.

**Keywords:** Dynamic graphs · Agents · Gathering


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 339–355, 2017.
https://doi.org/10.1007/978-3-319-72050-0_20

# 1 Introduction

## 1.1 Background and Problem

The *gathering* problem requires a set of $k$ mobile computational entities, dispersed at different locations in the spacial universe they inhabit, to group within finite time at the same location, not fixed in advanced. This problem, known also as *multi-agent rendezvous*, has been intesively and extensively studied in a variety of fields, including operations research (e.g., [1]) and control (e.g., [36]), the original focus being on the *rendezvous* problem, i.e. the special case $k = 2$.

In distributed computing, this problem has been extensively studied both in continuous and in discrete domains. In *continuous* domains, both gathering and rendevous have been investigated in the context of swarms of autonomous mobile *robots* operating in one- and two-dimensional spaces, requiring them to meet at (or converge to) the same point (e.g., see [10,11,16,24,25,37]). In *discrete* domains, the mobile entities, usually called *agents*, are dispersed in a network modeled as a graph and are required to gather at the same node (or at the two sides of the same edge) and terminate (e.g., see [2,17,18,21,22,29–32,40,41]). The main obstacle for solving the problem is *symmetry*, which can occur at several levels (topological structure, nodes, agents, communication), each playing a key role in the difficulty of the problem and of its resolution. For example, when the nodes are uniquely numbered, solving gathering is trivial. On the other hand, when the nodes are anonymous, the network is highly symmetric, the agents are identical, and there is no means of communication, the problem is clearly impossible to solve by deterministic means. The quest has been for minimal empowering assumptions which would make the problems deterministically solvable. A very common assumption is for the agents to have distinct *identities*; this enables different agents to execute different deterministic algorithms (e.g., see [12,17,18,41]). An alternative type of assumption consists in empowering the agents with some minimal form of *explicit communication*. In one approach, this is achieved by having a whiteboard at each node giving the agents the ability to leave notes in each node they travel (e.g., [2,9,21]). A less explicit and more primitive form of communication is by endowing each agent with a constant number of movable tokens, i.e. pebbles that can be placed on nodes, picked up, and carried while moving (e.g., [13]). An assumption much less demanding than agents having identities or explicit communication is that of having the homebases (i.e., the nodes where the agents are initially located) identifiable by an identical mark visible to any agent passing by it; originally suggested in [3], it has been used and studied e.g., in [22,32,39]. Summarizing, the existing literature on gathering and rendezvous is extensive and the variety of assumptions and results is aboundant (for surveys see [31,38]). Regardless of their differences, all these investigations, including those that consider faulty nodes (e.g., see [6,9,21]), share the same fundamental assumption that the topological structure does not change during the rendezvous or the gathering. In other words, they only consider *static graphs*.

Recently, within distributed computing, researchers started to investigate *dynamic graphs*, that is graphs where the topological changes are not localized and sporadic; rather, they occur continuously and at unpredictable locations, and are integral part of the nature of the system [8,35]. The study of distributed computations in dynamic graphs has concentrated on problems of information diffusion, agreement, and exploration (e.g., [4,5,7,23,26–28,33,34]).

In this paper we start the investigation of gathering in dynamic graphs by studying the feasibility of this problem in *dynamic rings*. Note that rendezvous and gathering in a ring, the prototypical symmetric graph, have been intesively studied in the static case (e.g., see the monograph on the subject [31]). The presence, in the static case, of a mobile faulty agent that can block other agents, considered in [14,15], could be seen as inducing a particular form of dynamics. Other than that, nothing is known on gathering in dynamic rings.

### 1.2 Main Contributions

We study gathering of $k$ agents, identical and without communication capabilities, in a dynamic ring of $n$ anonymous nodes with identically marked homebases. The class of dynamics we consider is the classic 1-interval-connectivity (e.g., [19,26,28,33,34]); that is, the system is fully synchronous and under a (possibly unfair) adversarial schedule that, at each round, chooses which edge (if any) will be missing. In this setting, we investigate under what conditions the gathering problem is solvable. In particular, we focus on the impact that factors such as *chirality* (i.e., common sense of orientation) and *cross detection* (i.e., the ability to detect, when traversing an edge, whether some agent is traversing it in the other direction), have on the solvability of the problem. Since, as we prove, gathering at a single node cannot be guaranteed in a dynamic ring, we allow gathering to occur either at the same node, or at the two end nodes of the same link.

A main result of our investigation is the complete characterization of the classes $\mathcal{F}(X,Y)$ of initial configurations from which the gathering problem is solvable with respect to chirality ($X \in \{\texttt{chirality}, \neg\texttt{chirality}\}$) and cross detection ($Y \in \{\texttt{detection}, \neg\texttt{detection}\}$). In obtaining this characterization, we establish several interesting results. For example, we show that, without chirality, cross detection is a powerful computational element; in fact, we prove (Theorems 1 and 5): $\mathcal{F}(\neg\texttt{chirality}, \neg\texttt{detection}) \subsetneq \mathcal{F}(\neg\texttt{chirality}, \texttt{detection})$. Furthermore, in such systems knowledge of the ring size $n$ cannot be substituted by knowledge of the number of agents $k$ (at least one of $n$ and $k$ must be known for gathering to be possible); in fact, we prove that, with cross detection but without chirality, knowledge of $n$ is strictly more powerful than knowledge of $k$. On the other hand, we show that, with chirality, knowledge of $n$ can be substituted by knowledge of $k$, yielding the same classes of feasible initial configurations. Furthermore, with chirality, cross detection is no longer a computational separator; in fact (Theorems 3 and 4) $\mathcal{F}(\texttt{chirality}, \neg\texttt{detection}) = \mathcal{F}(\texttt{chirality}, \texttt{detection})$. We also observe that $\mathcal{F}_{static} = \mathcal{F}(\texttt{chirality}, *) = \mathcal{F}(\neg\texttt{chirality}, \texttt{detection})$ where $\mathcal{F}_{static}$ denotes

the set of initial configurations from which gathering is possible in the static case. In other words: *with chirality or with cross detection, it is possible to overcome the computational obstacles created by the highly dynamic nature of the system.* All the feasibility results of this characterization are constructive: for each situation, we provide a distributed algorithm that allows the agents to gather within low polynomial time. In particular, the protocols for gathering with cross detection, terminating in $O(n)$ time, are time *optimal*. Moreover, our algorithms are *effective*; that is, starting from any arbitrary configuration $C$ in a ring with conditions $X$ and $Y$, within finite time the agents determine whether or not $C \in \mathcal{F}(X, Y)$ is feasible, and gather if it is.

Due to space limitations, the proofs are omitted; for the full text see [20].

## 2 Model and Basic Limitations

### 2.1 Model and Terminology

Let $\mathcal{R} = (v_0, \ldots v_{n-1})$ be a synchronous dynamic ring where, at any round $t \in N$, one of its edges might not be present; the choice of which edge is missing (if any) is controlled by an adversarial scheduler, not restricted by fairness assumptions. Such a dynamic network is known in the literature as a *1-interval connected* ring (e.g., [20,28]). Each node $v_i$ is connected to its two neighbours $v_{i-1}$ and $v_{i+1}$ via distinctly labeled ports $q_{i-}$ and $q_{i+}$, respectively (all operations on the indices are modulo $n$); the labels of the ports are arbitrary elements of a totally ordered set, and thus might not provide a globally consistent orientation. Each port of $v_i$ has an *incoming* buffer and an *outgoing* buffer. Finally, the nodes are *anonymous* (i.e., have no distinguished identifiers). Operating in $\mathcal{R}$ is a set $\mathcal{A} = \{a_0, \ldots, a_{k-1}\}$ of computational entities, called agents, each provided with memory and computational capabilities. The agents are *anonymous* (i.e., without distinguishing identifiers) and all execute the same protocol. When in a node $v$, an agent can be *at* $v$ or in one of the port buffers. Any number of agents can be in a node at the same time; an agent can determine how many other agents are in its location and where (in incoming buffer, in outgoing buffer, at the node). Initially the agents are located at distinct locations, called homebases; homebases are marked so that an agent can determine whether or not the current node is a homebase. Note that, as discussed later, this assumption is necessary in our setting. Each agent has its own *left*/*right* orientation of the ring, but the orientations of the agents might not be the same. If all agents agree on the orientation, we say that there is *chirality*. The agents are *silent*: they not have any explicit communication mechanism. They are *mobile*; that is, they can move from node to neighboring node. More than one agent may move on the same edge in the same direction in the same round. We say that the system has *cross detection* if whenever two or more agents move in opposite directions on the same edge in the same round, the involved agents detect this event; however they do not necessarily know the number of the involved agents in either direction. In each round, every agent is in one of a finite set of system states $\mathcal{S}$ which includes two special states: the initial state Init and the terminal state Term. At

the beginning of a round $r$, an agent in $v$ executes its protocol (the same for all agents). Based on the number of agents at $v$ and in its buffers, and on the content of its local memory and its state, the agent determines whether or not to move and, if so, in which direction ($direction \in \{left, right, nil\}$). If $direction = nil$, the agent places itself at $v$ (if currently on a port). If $direction \neq nil$, the agent moves in the outgoing buffer of the corresponding port (if not already there); if the link is present, it arrives in the incoming buffer of the destination node in round $r + 1$; otherwise it does not leave the outgoing buffer. As a consequence, an agent can be in an outgoing buffer at the beginning of a round only when the corresponding link was not present in the previous round. In the following, when an agents is in an outgoing buffer that leads to the missing edge, we will say that the agent is *blocked*. When multiple agents are at the same node, all of them have the same direction of movement, and are in the same state, we say that they form a *group*. Let $(\mathcal{R}, \mathcal{A})$ denote a system so defined. In $(\mathcal{R}, \mathcal{A})$, gathering is achieved in round $r$ if all agents in $A$ are on the same node or on two neighbouring nodes in $r$; in the first case, gathering is said to be *strict*. An algorithm *solves* GATHERING if, starting from any configuration from which gathering is possible, within finite time all agents are in the terminal state, are gathered, and are aware that gathering has been achieved. A solution algorithm is *effective* if starting from any configuration from which gathering is *not* possible, within finite time all agents detect such impossibility.

### 2.2 Configurations and Elections

The locations of the $k$ home bases in the ring is called a *configuration*. Let $\mathcal{C}$ be the set of all possible configurations with $k$ agents. Let $h_0, \ldots, h_{k-1}$ denote the nodes corresponding to the marked homebases (in a clockwise order) in $C \in \mathcal{C}$. We shall indicate by $d_i$ ($0 \leq i \leq k-1$) the distance (i.e., number of edges) between $h_i$ and $h_{i+1}$ (all operations are modulo $k$). Let $\delta^{+j}$ denote the *inter-distance* sequence clockwise $\delta^{+j} =< d_j, d_{j+1} \ldots d_{j+k-1} >$, and let $\delta^{-j}$ denote the couter-clockwise sequence $\delta^{-j} =< d_{j-1} \ \ldots \ d_{j-(k-1)} >$. The unordered pair of inter-distance sequences $\delta^{+j}$ and $\delta^{-j}$ describes the configuration from the point of view of node $h_j$. A configuration is *periodic* with period $p$ (with $p|k$) if $\delta_i = \delta_{i+p}$ for all $i = 0, \ldots k-1$. Let $\mathcal{P}$ denote the set of periodic configurations. Let $\Delta^+ = \{\delta^{+j} : 0 \leq j < k-1\}$ and $\Delta^- = \{\delta^{-j} : 0 \leq j < k-1\}$. We will denote by $\delta_{min}$ the ascending lexicographically minimum sequence in $\Delta^+ \cup \Delta^-$. Among the non-periodic configurations, particular ones are the *double-palindrome* configurations, where $\delta_{min} = \delta^{+i} = \delta^{-j}$ with $i \neq j$, where it is easy to see that the two sequences between the corresponding home bases $h_i$ and $h_j$ are both palindrome. A double-palindrome configuration has thus a unique axis of symmetry, equidistant from $h_i$ and $h_j$. If such an axis passes through two edges (i.e., the distances between $h_i$ and $h_j$ are both odd), we say that the configuration is *edge-edge*, and we denote by $\mathcal{E}$ the set of edge-edge configurations.

A characterization of the configurations where a leader can be elected depending on chirality is well known in static rings.

*Property 1.* In a static ring without chirality, a leader node can be elected from configuration $C$ if and only if $C \in \mathcal{C} \setminus (\mathcal{P} \cup \mathcal{E})$; a leader edge can be elected if and only if $C \in \mathcal{C} \setminus \mathcal{P}$.

With chirality, a leader node can be elected if and only if $C \in \mathcal{C} \setminus \mathcal{P}$.

### 2.3 Basic Limitations and Properties

The simple properties below motivate the necessity of the following assumptions: identical but distinguishable homebases, knowledge of either $n$ or $k$, gathering on a node or on an edge.

*Property 2.* If the homebases are not distinguishable, then GATHERING is unsolvable in $(\mathcal{R}, \mathcal{A})$; this holds regardless of chirality, cross detection, and knowledge of $k$ and $n$.

*Property 3.* In $(\mathcal{R}, \mathcal{A})$, if neither $n$ nor $k$ are known, then GATHERING is unsolvable; this holds regardless of chirality and cross detection.

*Property 4.* In $(\mathcal{R}, \mathcal{A})$, strict GATHERING is unsolvable; this holds regardless of chirality, cross detection, and knowledge of $k$ and $n$.

Finally, the following obvious but important limitation holds even in static situations.

*Property 5.* GATHERING is unsolvable if the initial configuration $C \in \mathcal{P}$; this holds regardless of chirality, cross detection, and knowledge of $k$ and $n$.

## 3 General Solution Structure

Our algorithms have the same general structure, and use the same building block and variables.

**General Structure.** All our algorithms are divided in two phases. The goal of Phase 1 is for the agents to explore the ring. In doing so, they may happen to solve GATHERING as well. If they complete Phase 1 without gathering, the agents are able to elect a node or an edge (depending on the specific situation) and the algorithm proceeds to Phase 2. In Phase 2 the agents try to gather around the elected node (or edge); however, gathering on that node (or edge) might not be possible due to the fact that the ring is dynamic. Different strategies are devised, depending on the setting, to guarantee that in finite time the problem is solved in spite of the choice of schedule of missing links decided by the adversary. For each setting, we will describe the two phases depending on the availability or lack of cross detection, as well as on the presence or not of chirality. Intuitively, cross detection is useful to simplify termination in Phase 2, chirality helps in breaking symmetries.

**Exploration Building Block.** At each round, an agent evaluates a set of predicates: depending on the result of this evaluation, it chooses a direction of

movement and possibly a new state. In its most general form, the evaluation of the predicates occurs through the building block procedure Explore ($dir$ | $p_1 : s_1$; $p_2 : s_2$; ...; $p_h : s_h$), where $dir$ is either *left* or *right*, $p_i$ is a predicate, and $s_i$ is a state. In Procedure Explore, the agent evaluates the predicates $p_1, \ldots, p_h$ in order; as soon as a predicate is satisfied, say $p_i$, the procedure exits and the agent does a transition to the specified state, say $s_i$. If no predicate is satisfied, the agent tries to move in the specified direction $dir$ and the procedure

**Table 1.** List of variables used in our algorithms.

| Variables | Description |
|---|---|
| $r_{ms}$ | It stores the last round when the agent meets someone (at a node) that is moving in the same direction (initially set to 0); this value is updated each time a new agent is met, and it is reset at each change of state or direction of movement |
| *Btime* | The number of rounds the executing agent has been blocked trying to traverse a missing edge since $r_{ms}$. This variable is reset to 0 each time the agent either traverses an edge or changes direction to traverse a new edge |
| *Etime*, *Esteps* | The total number of rounds and edge traversals, respectively. These values are reset at each new call of procedure Explore or when $r_{ms}$ is set |
| *Agents* | The number of agents at the node of the executing agent. This value is set at each round |

**Table 2.** List of predicates used in our algorithms.

| Predicate | Description |
|---|---|
| *meeting* | Satisfied when the agent (either in a port or at a node) detects an increase in the numbers of agents it sees at each round |
| *meetingSameDir* | Satisfied when the agent detects, in the current round, new agents moving in its same direction. This is done by seeing new agents in an incoming or outgoing buffer corresponding to a direction that is equal to the current direction of the agent |
| *meetingOppositeDir* | Satisfied when the agent detects, in the current round, new agents moving in its opposite direction. This is done by seeing new agents in an incoming or outgoing buffer corresponding to a direction that is opposite to the current direction of the agent |
| *crossed* | Satisfied when the agent, while traversing a link, detects in the current round other agent(s) moving on the same link in the opposite direction |
| *seeElected* | Let us assume there is either an elected node or an elected edge. This predicate is satisfied when the agent has reached the elected node or an endpoint of the elected edge |

is executed again in the next round. Predicates and variables used by procedure Explore are indicated in Tables 1 and 2.

## 4 Gathering with Cross Detection

In this section, we study gathering in dynamic rings when there is cross detection; that is, an agent crossing a link can detect whether other agents are crossing it in the opposite direction. Recall that, by Property 3, at least one of $n$ and $k$ must be known. We first examine the problem without chirality and show that, with knowledge of $n$, it is sovable in all configurations that are feasible in the static case; furthermore, this is done in optimal time $\Theta(n)$. On the other hand, with knowledge of $k$ alone, the problem is unsolvable. We then examine the problem with chirality, and show that in this case the problem is sovable in all configurations that are feasible in the static case even with knowledge of $k$ alone; furthermore, this is done in optimal time $\Theta(n)$.

### 4.1 With Cross Detection: Without Chirality

In this section, we present and analyze the algorithm, Gather(Cross, $\not\subset$hir), that solves Gathering in rings of known size with cross setection but without chirality.

**Algorithm** Gather(Cross,$\not\subset$hir)**: Phase 1.** The overall idea of this phase, shown in Fig. 1, is to let the agents move long enough along the ring to guarantee that, if they do not gather, they all manage to fully traverse the ring in spite of the link removals. More precisely, for the first $6n$ rounds each agent attempts to move to the left (according to its orientation). At round $6n$, the agent checks if the predicate $Pred \equiv (r_{ms} < 3n \wedge Esteps < n)$ is verified. If *Pred* is not verified, then (as we show) the agent has explored the entire ring and thus knows the total number $k$ of agents (local variable $TotalAgents$); in this case, the agent switches direction, and enters state SwitchDir. Otherwise, if after $6n$ rounds *Pred* is satisfied, then $k$ is not known yet: in this case, the agent keeps the same direction, and enters state KeepDir. In state SwitchDir, the agent attempts to move in the chosen direction until round $12n$. At round $12n$, the agent terminates if the predicate $[r_{ms} < 9n \wedge Esteps < n]$ holds, predicate *meetingOppositeDir* does not hold, and in its current node there are $k$ agents; otherwise, it starts Phase 2. In state KeepDir, if at any round predicate *crossed* or predicate *meetingOppositeDir* hold, the agent terminates; otherwise, it attempts to move to its left until round $12n$. At round $12n$, if the predicate $[r_{ms} < 9n \wedge Esteps < n]$ holds, the agent terminates; otherwise, it switches to Phase 2.

We now prove some important properties of Phase 1.

**Lemma 1.** *Let agent $a^*$ move less than $n$ steps in the first $3n$ rounds. Then, by round $3n$, all agents moving in the same direction as $a^*$ belong to the same group.*

States: {Init, SwitchDir, KeepDir, Term}.
In state Init:
EXPLORE ($left \mid Ttime = 6n \ \wedge \ \neg Pred$: SwitchDir; $Ttime = 6n \ \wedge \ Pred$: KeepDir)
In state SwitchDir:
EXPLORE($right \mid Ttime = 12n \wedge r_{ms} < 9n \ \wedge \ Esteps < n \ \wedge \ Agents = TotalAgents \wedge \neg meetingOppositeDir$: Term; $Ttime = 12n$: Phase 2)
In state KeepDir:
EXPLORE ($left \mid crossed \vee meetingOppositeDir$: Term; $Ttime = 12n \ \wedge \ r_{ms} < 9n \ \wedge \ Esteps < n$: Term; $Ttime = 12n$: Phase 2)

**Fig. 1.** Phase 1 of Algorithm GATHER(CROSS, $\not\subset$HIR)

Because of absence of chirality, the set $\mathcal{A}$ of agents can be partitioned into two sets where all the agents in the same set share the same orientation of the ring; let $A_r$ and $A_l$ be the two sets.

**Lemma 2.** *Let $A \in \{A_r, A_l\}$. If at round $6n$* Pred *is verified for an agent $a^* \in A$, then all agents in $A$ are in the same group at round $6n$. Moreover,* Pred *is verified for all agents in $A$.*

**Lemma 3.** *Let $A \in \{A_r, A_l\}$. If* Pred *is not verified at round $6n$ for agent $a^* \in A$, then at round $6n$ all agents in $A$ have done a complete tour of the ring (and hence know the number of total agents, $k$); moreover,* Pred *is not verified for all agents in $A$.*

**Lemma 4.** *If one agent terminates in* Phase 1*, then all agents terminate and gathering has been achieved. Otherwise, no agent terminates and all of them have done a complete tour of the ring.*

**Algorithm** GATHER(CROSS,$\not\subset$HIR)**: Phase 2.** When the agents execute Phase 2, by Lemma 4, they know the initial configuration $C$. If $C \in \mathcal{P}$, gathering is impossible (Property 5) and they become aware of this fact. Otherwise, if $C \in \mathcal{E}$ they can elect an edge $e_L$, and if $C \in \mathcal{C} \setminus (\mathcal{P} \cup \mathcal{E})$ they can elect one leader node $v_L$ (Property 1). For simplicity of exposition and w.l.g., in the following we assume that Phase 2 of the algorithm starts at round 0. In Phase 2, an agent first resets all its local variables, with the exception of $TotalAgents$, that stores the number of agents $k$; between rounds 0 and $3n$, each agent moves toward the elected edge/node following the shortest path (shortestPathDirectionElected()). If by round $3n$ an agent has reached the elected node or an endpoint of the elected edge it stops, and enters the ReachedElected state. Otherwise (i.e., at round $3n$, the agent is not in state ReachedElected), it switches to the ReachingElected state. If all agents are in the same state (either ReachedElected or ReachingElected), then they are in the same group, and terminate ($Agents = TotalAgents$). If they do not terminate, all agents start moving: each ReachingElected agent in the same direction it chose at the beginning of Phase 2, while the ReachedElected agents reverse direction. From this moment, each agent, regardless of its state,

terminates as soon as it perceives $k$ agents in the same node or if it is blocked on a missing edge for $2n$ rounds. In other situations, the behaviour of agent $a^*$ depends on its state, as described below.

**State ReachedElected**. If $a^*$ crosses a group of agents, it enters the Joining state. In this new state, say at node $v$, the agent switches direction in the attempt to catch and join the agent(s) it just crossed. If $a^*$ leaves $v$ without crossing any agent ($Esteps = 1$), $a^*$ enters again the ReachedElected state, switching again direction (i.e., it goes back to direction originally chosen when Phase 2 started). If instead $a^*$ leaves $v$ and it crosses some agents, it terminates: this can happen because also the agents that $a^*$ crossed try to catch it (and all other agents in the same group with $a^*$). As we will show, in this case all agents can correctly terminate.

**State ReachingElected**. If $a^*$ is able to reach the elected node/edge (*seeElected* is verified), it enters the ReachedElected state, and switches direction. If $a^*$ is blocked on a missing edge and it is reached by other agents, then it switches state to ReachedElected keeping its direction (*meetingSameDir* is verified). Finally, if $a^*$ crosses someone, it enters the Waiting state, and it stops moving. If while in the Waiting state $a^*$ meets someone new before $2n$ rounds, it enters the ReachedElected state, and switches direction. Otherwise, at round $2n$ it terminates (Fig. 2).

States: {Phase 2, ReachedElected, ReachingElected, Joining, Waiting, ReverseDir,Term}.
In state Phase 2:
  **if** $C \in \mathcal{P}$ **then**
    unsolvable()
    Go to State Term
  resetAllVariables except $TotalAgents$
  $dir$ =shortestPathDirectionElected()
  EXPLORE ($dir$ | *seeElected*: ReachedElected; $Ttime = 3n$: ReachingElected)
In state ReachedElected:
  $dir$ =opposite(dir)
  **if** $Ttime \geq 3n$ **then**
    EXPLORE ($dir$ | $Agents = TotalAgents \vee Btime = 2n$: Term; *crossed*: Joining)
In state Joining:
  $dir$ =opposite(dir)
  EXPLORE ($dir$ | $Agents = TotalAgents \vee Btime = 2n \vee$ *crossed*: Term; $Esteps = 1$: ReverseDir)
In state ReachingElected:
  EXPLORE ($dir$ | $Agents = TotalAgents \vee Btime = 2n$: Term; *meetingSameDir*: ReachedElected; *meetingOppositeDir* $\vee$ *seeElected*: ReverseDir; *crossed*: Waiting)
In state Waiting:
  EXPLORE (nil | $Etime > 2n$: Term; *meeting*: ReverseDir)
In state ReverseDir:
  $dir$ =opposite(dir)
  Go to State ReachedElected

**Fig. 2.** Phase 2 of Algorithm GATHER(CROSS, $\emptyset$CHIR)

**Lemma 5.** *At round* $3n$ *of* Phase 2*, there is at most one group of agents in state* ReachingElected*, and at most two groups of agents in state* ReachedElected*.*

**Lemma 6.** *If an agent* $a^*$ *terminates executing* Phase 2*, then all other agents will terminate, and gathering is correctly achieved.*

**Lemma 7.** Phase 2 *terminates in at most* $10n$ *rounds.*

**Theorem 1.** *Without chirality,* Gathering *is solvable in rings of known size with cross detection, starting from any* $C \in \mathcal{C} \setminus \mathcal{P}$*. This can be done in* $\mathcal{O}(n)$ *rounds by an effective algorithm.*

### 4.2 Knowledge of $n$ is More Powerful Than Knowledge of $k$

One may ask if it is possible to obtain the same result of Theorem 1 if knowledge of $k$ was available instead of $n$; recall that at least one of $n$ and $k$ must be known (Property 3). Unfortunately, the following Theorem shows that, from a computational point of view, knowledge of the ring size is strictly more powerful than knowledge of the number of agents.

**Theorem 2.** *In rings with no chirality,* Gathering *is impossible without knowledge of* $n$ *when starting from a configuration* $C \in \mathcal{E}$*. This holds even if there is cross detection and* $k$ *is known.*

### 4.3 With Cross Detection: With Chirality

Let us now consider the simplest setting, where the agents have cross detection capability as well as a common chirality. If $n$ is known, the problem is already optimally solved by Algorithm Gather(Cross, $\not\subset$hir). So, we need only to consider the case when $k$ is known but $n$ is not.

Phase 1 of the solution, Algorithm Gather(Cross, Chir), consists of the following modification of Phase 1 of Algorithm Gather(Cross, $\not\subset$hir), to work with knowledge of $k$. Each agent moves counterclockwise terminating if the $k$ agents are all at the same node. As soon as it passes by $k+1$ homebases, it discovers $n$. At this point, it continues to attempt to move in the same direction switching to Phase 2 at round $3n+1$ (unless gathering occurs before). After $3n$ rounds, if the agents have not terminated, they have however certainly performed a loop of the ring, know $n$ (having seen $k+1$ home bases) and they start Phase 2. Since $n$ is known, the agents can use as Phase 2 the one of Algorithm Gather(Cross, $\not\subset$hir), which is time optimal. We then have:

**Theorem 3.** *With chirality, cross detection and knowledge of either* $n$ *or* $k$*,* Gathering *is solvable in at most* $\mathcal{O}(n)$ *rounds from any configuration* $C \in \mathcal{C} \setminus \mathcal{P}$ *with an effective algorithm..*

## 5 Without Cross Detection

In this section we study the gathering problem when there is no cross detection. We focus first on the case when the absence of cross detection is mitigated by the presence of chirality. We show that gathering is possible in the same class of configurations as with cross detection, albeit with a $O(n \log n)$ time complexity. We then examine the most difficult case of absence of both cross detection and chirality. We prove that in this case the class of feasible configurations is smaller (i.e., cross detection is a computational separator). We show that gathering can be performed from all feasible configuration in $O(n^2)$ time.

### 5.1 Without Cross Detection: With Chirality

The structure of the algorithm, Gather(⌀Cross,Chir), still follows the two Phases. Since Phase 1 of Algorithm Gather(Cross,Chir) does not use cross detection, it can be used as Phase 1 of Gather(⌀Cross,Chir).

Let thus focus on Phase 2. Because of chirality, a leader node can be always elected, even when the initial configuration is in $\mathcal{E}$ (Property 1). We will show how to use this fact to modify Phase 2 of Algorithm Gather(Cross,Chir) to work without assuming cross detection. We will do so by designing a mechanism that will force the agents *never to cross each other*. The main consequence of this fact is that, whenever two agents (or two groups of agents) would like to traverse the same edge in opposite direction, only one of the two will be allowed to move thus "merging" with the other. This mechanism is described below.

**Basic no-crossing mechanism.** To avoid crossings, each agent constructs an edge labeled bidirectional directed ring with $n$ nodes (called *Logic Ring*) and it moves on the actual ring according to the algorithm, but also to specific conditions dictated by the labels of the *Logic Ring*. In the *Logic Ring*, each edge of the actual ring is replaced by two labeled oriented edges in the two directions. The label of each oriented edge $e_i$, $0 \leq i \leq n-1$, is either $X_i$ or $Y_i$ , where $X_i$ and $Y_i$ are infinite sets of integers. Labels $X_0 \dots X_{n-1}$ are assigned to consecutive edges in counter-clockwise direction starting from the leader node, while $Y_0 \dots Y_{n-1}$ are assigned in clockwise direction. Intuitively, we want to construct these sets of labels in such a way that $X_i$ and $Y_i$ have an empty intersection, and allow an agent to traverse an edge at round $r$ only if $r$ is contained in the set of labels associated to the corresponding oriented edge of the *Logic Ring*. For this construction we define $X_i = \{s + m \cdot (2p+2) \mid (s \in S_i \vee s = 2p), \forall m \in \mathbb{N}\}$, where $p = \lceil \log_2 n \rceil$, and $S_i$ is a subset of $\{0, 1, \dots, 2p-1\}$ of size exactly $p$ (note that there are $\binom{2p}{p} \geq n$ possible choices for $S_i$). Indeed, there are $2^p = 2^{\lceil \log_2 n \rceil} \geq n$ ways to choose which elements of $\{0, 1, \dots, p-1\}$ are in $S_i$; each of these choices can be completed to a set of size $p$ by choosing the remaining elements from the set $\{p, p+1, \dots, 2p-1\}$. Therefore there are at least $n$ available labels, and we can define the $X_i$'s so that they are all distinct. Then we define $Y_i$ to be the complement of $X_i$ for every $i$. The following property is immediate by construction:

States: {ReachedElected, ReachingElected, ChangeDir, ChangeState, DirCommR, DirCommS, Term}.
In state Phase 2:
  **if** $C \in \mathcal{P}$ **then**
    unsolvable()
    Go to State Term
  resetAllVariables except $TotalAgents$
  $dir$ = leaderMinimumPath()
  EXPLORE ($dir$ | $seeElected$: ReachedElected; $Ttime = 3n$: ReachingElected)
In state ReachedElected:
  **if** $Ttime \geq 3n$ **then**
    $dir$ = clockwiseDirection()
    EXPLORE ($dir$ | ($BPeriods \geq 4n + 8 \ \vee \ Agents = TotalAgents$): Term;)
In state ReachingElected:
  **if** $Ttime = 3n$ **then**
    $dir$ = counterclockwiseDirection()
  EXPLORE (dir | ($BPeriods \geq 4n + 8 \ \vee \ Agents = TotalAgents$): Term;)

**Fig. 3.** Phase 2 of Algorithm GATHER(CROSS,CHIR)

**Observation 1.** *Let $m \in \mathbb{N}$ and let $I = \{m, m+1, \ldots, m+2p+1\}$. Then, $X_i$ and $Y_j$ have a non-empty intersection in $I$ if and only if $i \neq j$, $X_i$ and $X_j$ have a non-empty intersection in $I$, and $Y_i$ and $Y_j$ have a non-empty intersection in $I$.*

From the previous observation, it follows that two agents moving following the *Logic Ring* in opposite directions will never cross each other on an edge of the actual ring. As a consequence of this fact, we can derive a bound on the number of rounds that guarantee two groups of robots moving in opposite direction, to "merge". In the following lemma, we consider the execution of the algorithm proceeding in *periods*, where each period is composed by $2p+2$ rounds. We have:

**Lemma 8.** *Let us consider two groups of agents, $G$ and $G'$, moving in opposite directions following the* Logic Ring*. After at most $n$ periods, that is at most $\mathcal{O}(n \log n)$ rounds, the groups will be at a distance $d \leq 1$ (in the direction of their movements).*

We are now ready to describe the actual Phase 2. In the following, when the agents are moving following the meta-rule in the *Logic Ring*, we will use variable $BPeriods$, instead of $Btime$, indicating the number of consecutive periods in which the agent failed to traverse the current edge. As in the case of $Btime$, the new variable $BPeriods$ is reset each time the agent traverses the edge, changes direction, or encounters new agents in its moving direction. In the first $3n$ rounds, each agent moves towards the elected node using the minimum distance path. After round $3n$, on the logic ring, the group in state ReachedElected starts moving in clockwise direction, the group in state ReachingElected in counterclockwise. One of the two groups terminates if $BPeriods \geq n$ rounds or if $Agents = k$. This replaces the terminating condition $Btime = 2n$ that was used in case of Cross detection. Phase 2 of the Algorithm is shown in Fig. 3.

**Lemma 9.** *Phase 2 of Algorithm* GATHER(CROSS,CHIR) *terminates in at most $\mathcal{O}(n \log n)$ rounds, solving the* GATHERING *problem.*

**Theorem 4.** *With chirality and knowledge of $n$ or $k$,* GATHERING *is solvable from any configuration $C \in \mathcal{C} \setminus \mathcal{P}$. This can be done in $\mathcal{O}(n \log n)$ rounds with an effective algorithm.*

### 5.2 Without Cross Detection: Without Chirality

In this section, we consider the most difficult setting when neither cross detection nor chirality are available. We show that in this case GATHERING is impossible if $C \in \mathcal{E}$. On the other hand, we provide a solution for rings of known size from any initial configuration $C \in \mathcal{C} \setminus (\mathcal{P} \cup \mathcal{E})$, which works in $\mathcal{O}(n^2)$ rounds. We start this Section with the impossibility result.

**Theorem 5.** *Without chirality and without cross detection,* GATHERING *is impossible when starting from a configuration $C \in \mathcal{E}$. This holds even if the agents know $C$ (hence $n$ and $k$).*

**Algorithm** GATHER(∅CROSS,∅CHIR)**: Phase 1.** The lack of cross detection is not a problem when there is a common chirality. However, the combination of lack of both cross detection and chirality significantly complicates Phase 1, and new mechanisms have to be devised to insure that all agents finish the ring exploration and correctly switch to Phase 2. In the following we will denote by $Btime'$ the value of $Btime$ at the previous round, that is at round $Ttime - 1$.

Each agent attempts to move along the ring in its own left direction. An agent terminates in the Init state if it has been blocked long enough ($Btime \geq 2n+2$), or if it was blocked for an appropriate amount of time and is now meeting a new agent ($Btime' \geq n+1 \wedge meeting$). If an agent does not terminate by round $(3n)(n+3)$ it enters a *sync sub-phase*, whose purpose is to perform a synchronization to ensure that, if a group of agents terminates in the Init state by condition ($Btime \geq 2n+2$), all the remaining active agents will also terminate correctly (Fig. 4).

States: {Init, SyncR, SyncL, Term}.
In state Init:
EXPLORE ($left \mid Ttime \geq (3n)(n+3)$: SyncL; $Btime \geq (2n+2) \vee (Btime' \geq n+1 \wedge meeting)$: Term)
In state SyncL:
EXPLORE ($left \mid (Ttime \geq (3n)(n+3) + 2n + 1 \wedge Btime > n) \vee Agents = TotalAgents$: Term; $Ttime \geq (3n)(n+3) + 2n + 1$: Phase 2; $0 < Btime \leq n$: SyncR)
In state SyncR:
EXPLORE ($right \mid Agents = TotalAgents$: Term; $Ttime \geq (3n)(n+3) + 2n + 1$: Phase 2; $Btime = 1$: SyncL)

**Fig. 4.** Phase 1 of Algorithm GATHER(∅CROSS, ∄CHIR)

**Lemma 10.** *If an agent does not terminate at the end of* Phase 1*, then no agent terminates and all of them have done at least one complete loop of the ring. If an agent terminates during* Phase 1*, then all agents terminate and* Gathering *is correctly solved.*

**Algorithm** Gather($\not\subset$ross,$\not\subset$hir)**: Phase 2.** By Lemma 10, at the end of Phase 1 each agent knows the current configuration. Since we know that the problem is not solvable for initial configurations $\mathcal{C} \in \mathcal{E}$ (Theorem 5), the initial configuration must be non-symmetric (i.e., without any axis of symmetry) or symmetric but with the unique axis of symmetry going through a node. In both cases, the agents can agree on a common chirality. In fact, if $\mathcal{C}$ does not have any symmetry axes, the agents can agree, for example, on the direction of the lexicographically smallest sequence of homebases inter distances. If instead there is an axis of symmetry going through a node $v_L$, they can agree on the direction of the port of $v_L$ with the smallest label. We can then use as Phase 2, the one of Algorithm Gather($\not\subset$ross, Chir) presented in Sect. 5.1.

**Theorem 6.** *Without chirality,* Gathering *is solvable in rings of known size without cross detection from all* $C \in \mathcal{C} \setminus (\mathcal{P} \cup \mathcal{E})$*. This can be done in* $\mathcal{O}(n^2)$ *rounds by an effective algorithm.*

## References


1. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. Kluwer, Boston (2003). https://doi.org/10.1007/b100809
2. Barrière, L., Flocchini, P., Fraigniaud, P., Santoro, N.: Rendezvous and election of mobile agents: impact of sense of direction. Theor. Comput. Syst. **40**(2), 143–162 (2007)
3. Baston, V., Gal, S.: Rendezvous search when marks are left at the starting points. Naval Res. Logist. **38**, 469–494 (1991)
4. Biely, M., Robinson, P., Schmid, U., Schwarz, M., Winkler, K.: Gracefully degrading consensus and $k$-set agreement in directed dynamic networks. In: Bouajjani, A., Fauconnier, H. (eds.) NETYS 2015. LNCS, vol. 9466, pp. 109–124. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26850-7_8
5. Bournat, M., Datta, A.K., Dubois, S.: Self-stabilizing robots in highly dynamic environments. In: Bonakdarpour, B., Petit, F. (eds.) SSS 2016. LNCS, vol. 10083, pp. 54–69. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49259-9_5
6. Bouchard, S., Dieudonne, Y., Ducourthial, B.: Byzantine gathering in networks. Distrib. Comput. **29**(6), 435–457 (2016)
7. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Measuring temporal lags in delay-tolerant networks. IEEE Trans. Comput. **63**(2), 397–410 (2014)
8. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. Int. J. Parallel Emergent Distrib. Syst. **27**(5), 387–408 (2012)
9. Chalopin, J., Das, S., Santoro, N.: Rendezvous of mobile agents in unknown graphs with faulty links. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 108–122. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75142-7_11

10. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Distributed computing by mobile robots: gathering. SIAM J. Comput. **41**(4), 829–879 (2012)
11. Cohen, R., Peleg, D.: Convergence properties of the gravitational algorithm in asynchronous robot systems. SIAM J. Comput. **34**, 1516–1528 (2005)
12. Czyzowicz, J., Labourel, A., Pelc, A.: How to meet asynchronously (almost) everywhere. ACM Trans. Algorithms **8**(4), 37:1–37:14 (2012)
13. Czyzowicz, J., Dobrev, S., Kranakis, E., Krizanc, D.: The power of tokens: rendezvous and symmetry detection for two mobile agents in a ring. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 234–246. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77566-9_20
14. Das, S., Luccio, F.L., Focardi, R., Markou, E., Moro, D., Squarcina, M.: Gathering of robots in a ring with mobile faults. In: 17th Italian Conference on Theoretical Computer Science (ICTCS), pp. 122–135 (2016)
15. Das, S., Luccio, F.L., Markou, E.: Mobile agents rendezvous in spite of a malicious agent. In: Bose, P., Gąsieniec, L.A., Römer, K., Wattenhofer, R. (eds.) ALGOSENSORS 2015. LNCS, vol. 9536, pp. 211–224. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28472-9_16
16. Degener, B., Kempkes, B., Langner, T., Meyer auf der Heide, F., Pietrzyk, P., Wattenhofer, R.: A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In: 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 139–148 (2011)
17. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. Theoret. Comput. Sci. **355**, 315–326 (2006)
18. Dessmark, A., Fraigniaud, P., Pelc, A.: Deterministic rendezvous in graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 184–195. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39658-1_19
19. Di Luna, G.A., Dobrev, S., Flocchini, P., Santoro, N.: Live exploration of dynamic rings. In: 36th IEEE International Conference on Distributed Computing Systems, (ICDCS), pp. 570–579 (2016)
20. Di Luna, G.A., Flocchini, P., Pagli, L., Prencipe, G., Santoro, N., Viglietta, G.: Gathering in dynamic rings. Arxiv, April 2017
21. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Multiple agents RendezVous in a ring in spite of a black hole. In: Papatriantafilou, M., Hunel, P. (eds.) OPODIS 2003. LNCS, vol. 3144, pp. 34–46. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27860-3_6
22. Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Multiple Mobile Agent Rendezvous in a Ring. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 599–608. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24698-5_62
23. Flocchini, P., Mans, B., Santoro, N.: On the exploration of time-varying networks. Theoret. Comput. Sci. **469**, 53–68 (2013)
24. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theoret. Comput. Sci. **337**(1–3), 147–168 (2005)
25. Flocchini, P., Santoro, N., Viglietta, G., Yamashita, M.: Rendezvous with constant memory. Theoret. Comput. Sci. **621**, 57–72 (2016)
26. Haeupler, B., Kuhn, F.: Lower bounds on information dissemination in dynamic networks. In: Aguilera, M.K. (ed.) DISC 2012. LNCS, vol. 7611, pp. 166–180. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33651-5_12

27. Ilcinkas, D., Klasing, R., Wade, A.M.: Exploration of constantly connected dynamic graphs based on cactuses. In: Halldórsson, M.M. (ed.) SIROCCO 2014. LNCS, vol. 8576, pp. 250–262. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09620-9_20
28. Ilcinkas, D., Wade, A.M.: Exploration of the $T$-interval-connected dynamic graphs: the case of the ring. In: Moscibroda, T., Rescigno, A.A. (eds.) SIROCCO 2013. LNCS, vol. 8179, pp. 13–23. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03578-9_2
29. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theoret. Comput. Sci. **390**(1), 27–39 (2008)
30. Kranakis, E., Krizanc, D., Markou, E.: Mobile agent rendezvous in a synchronous torus. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 653–664. Springer, Heidelberg (2006). https://doi.org/10.1007/11682462_60
31. Kranakis, E., Krizanc, D., Markou, E.: The Mobile Agent Rendezvous Problem in the Ring. Morgan & Claypool (2010)
32. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous problem in the ring. In: 23rd International Conference on Distributed Computing Systems (ICDCS), pp. 592–599 (2003)
33. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: 42th Symposium on Theory of Computing (STOC), pp. 513–522 (2010)
34. Kuhn, F., Moses, Y., Oshman, R.: Coordinated consensus in dynamic networks. In: 30th Symposium on Principles of Distributed Computing (PODC), pp. 1–10 (2011)
35. Kuhn, F., Oshman, R.: Dynamic networks: models and algorithms. SIGACT News **42**(1), 82–96 (2011)
36. Lin, J., Morse, A.S., Anderson, B.D.O.: The multi-agent rendezvous problem. Parts 1 and 2. SIAM J. Control Optim. **46**(6), 2096–2147 (2007)
37. Pagli, L., Prencipe, G., Viglietta, G.: Getting close without touching: near-gathering for autonomous mobile robots. Distrib. Comput. **28**(5), 333–349 (2015)
38. Pelc, A.: Deterministic rendezvous in networks: a comprehensive survey. Networks **59**(3), 331–347 (2012)
39. Sawchuk, C.: Mobile agent rendezvous in the ring. Ph.D Thesis, Carleton University, January 2004
40. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. ACM Trans. Algorithms **10**(3), 12 (2014)
41. Yu, X., Yung, M.: Agent rendezvous: a dynamic symmetry-breaking problem. In: Meyer, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 610–621. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61440-0_163

# On Liveness of Dynamic Storage

Alexander Spiegelman(✉) and Idit Keidar

Viterbi EE Department, Technion, Haifa, Israel
{sashas,idish}@campus.technion.ac.il

**Abstract.** Dynamic distributed storage algorithms such as DynaStore, Reconfigurable Paxos, RAMBO, and RDS, do not ensure liveness (wait-freedom) in asynchronous runs with infinitely many reconfigurations. We prove that this is inherent for asynchronous dynamic storage algorithms. Our result holds even if only one process may fail, provided that machines that were successfully removed from the system's configuration can be switched off by a system administrator. To circumvent this result, we define a dynamic eventually perfect failure detector, and present an algorithm that uses it to emulate wait-free dynamic atomic storage. Though some of the previous algorithms have been designed for eventually synchronous models, to the best of our knowledge, our algorithm is the first to ensure liveness for all operations without restricting the reconfiguration rate.

## 1 Introduction

Many works in the last decade have dealt with *dynamic* reliable distributed storage emulation [2,5–9,13–15,17,18,21,25,27]. The motivation behind such storage is to allow new processes (nodes) to be phased in and old or dysfunctional ones to be taken offline. From a fault-tolerance point of view, once a faulty process is removed, additional failures may be tolerated. For example, consider a system that can tolerate one failure: once a process fails, no additional processes are allowed to fail. However, once the faulty process is replaced by a correct one, the system can again tolerate one failure. Thus, while static systems become permanently unavailable after some constant number of failures, dynamic systems that allow infinitely many reconfigurations can survive forever.

Previous works can be categorized into two main types: Solutions of the first type assume a churn-based model [19,24] in which processes are free to announce when they join the storage emulation [4–7] via an auxiliary broadcast sub-system that allows a process to send a message to all the processes in the system, (which my be unknown to the sending processes). The second type solutions extend the register's API with a reconfiguration operation for changing the current configuration of participating processes [2,9,13–15,18,25], which can be only invoked by members of the current configuration. In this paper we consider

A. Spiegelman is grateful to the Azrieli Foundation or the award of an Azrieli Fellowship.


S. Das and S. Tixeuil (Eds.): SIROCCO 2017, LNCS 10641, pp. 356–376, 2017.
https://doi.org/10.1007/978-3-319-72050-0_21

the latter. Such an API allows administrators (running privileged processes), to remove old or faulty processes and add new ones without shutting down the service; once a process is removed from the current configuration, a system administrator may shut it down. Note that in the churn-based model, in contrast, if processes have to perform an explicit operation in order to leave the system (as in [4,7]), a faulty process can never be removed. In addition, since in API-based models only processes that are already within the system invoke operations, it is possible to keep track of the processes in the system, and thus auxiliary broadcast is not required.

Though the literature is abundant with dynamic storage algorithms in both models, to the best of our knowledge, all previous solutions in asynchronous and eventually synchronous models restrict reconfigurations in some way in order to ensure completion of all operations. Churn-based solutions assume a bounded churn rate [4,5,7], meaning that there is a finite number of joining and removing processes in a given time interval. Some of the API-based solutions [2,13,18,25] provide liveness only when the number of reconfigurations is finite, whereas others discuss liveness only in synchronous runs [9,14,15]. Such restrictions may be problematic in emerging highly-dynamic large-scale settings.

Baldoni et al. [5] showed that it is impossible to emulate a dynamic register that ensures completion of all operations without restricting the churn rate in asynchronous churn-based models in which processes can freely abandon the computation without an explicit leave operation. Since a leave and a failure are indistinguishable in such models, the impossibility can be proven using a partition argument as in [3].

In this paper we revisit this question in the API-based model. First, we prove a similar result for asynchronous API-based dynamic models, in which *one* unremoved process can fail and successfully removed ones can go offline. Specifically, we show that even the weakest type of storage, namely a *safe* register [20], cannot be implemented so as to guarantee liveness for all operations (i.e., wait-freedom) in asynchronous runs with an unrestricted reconfiguration rate. Note that this bound does not follow from the one in [5] since a process in our model can leave the system only after an operation that removes it successfully completes.

Second, to circumvent our impossibility result, we define a dynamic failure detector that can be easily implemented in eventually synchronous systems, and use it to implement dynamic storage. We present an algorithm, based on state machine replication, that emulates a strong shared object, namely a wait-free atomic dynamic multi-writer, multi-reader (MWMR) register, and ensures liveness for all operations without restricting the reconfiguration rate. Though a number of previous algorithms have been designed for eventually synchronous models [5,7–9,14,15,21], to the best of our knowledge, our algorithm is the first to ensure liveness of all operations without restricting the reconfigurations rate.

In particular, previous algorithms [8,9,14,15,21] that used failure detectors, only did so for reaching consensus on the new configuration. For example, reconfigurable Paxos variants [8,21], which implement atomic storage via dynamic state machine replication, assume a failure detector that provides a leader in

every configuration. However, a configuration may be changed, allowing the previous leader to be removed (and then fail) before another process $p$ (with a pending operation) is able to communicate with it in the old configuration. Though a new leader is elected by the failure detector in the ensuing configuration, this scenario may repeat itself indefinitely, so that $p$'s pending operation never completes.

We, in contrast, use the failure detector also to implement a helping mechanism, which ensures that eventually some process will help a slow one before completing its own reconfiguration operation even if the reconfiguration rate is unbounded. Such mechanism is attainable in API-based models since only members of the current configuration invoke operations, and thus helping process can know which processes may need help. Note that in churn-based models in which processes announce their own join, implementing such a helping mechanism is impossible, since a helping process cannot possibly know which processes need help joining.

The remainder of this paper is organized as follows: In Sect. 2 we present the model and define the dynamic storage object we seek to implement. Our impossibility proof appears in Sect. 3, and our algorithm in Sect. 4. Finally, we conclude the paper in Sect. 5.

## 2 Model and Dynamic Storage Problem Definition

In Sect. 2.1, we present the preliminaries of our model, and in Sect. 2.2, we define the dynamic storage service.

### 2.1 Preliminaries

We consider an asynchronous message passing system consisting of an infinite set of processes $\Pi$. Processes may fail by crashing subject to restrictions given below. Process failure is modeled via an explicit fail action. Each pair of processes is connected by a communication link. A *service* exposes a set of *operations*. For example, a dynamic storage service exposes read, write, and reconfig operations. Operations are invoked and subsequently respond.

An *algorithm* $A$ defines the behaviors of processes as deterministic state machines, where state transitions are associated with *actions*, such as send/receive messages, operation invoke/response, and process failures. A *global state* is a mapping to states from system components, i.e., processes and links. An *initial global state* is one where all processes are in initial states and all links are empty. A send action is *enabled* in state $s$ if $A$ has a transition from $s$ in which the send occurs.

A *run* of algorithm $A$ is a (finite or infinite) alternating sequence of global states and actions, beginning with some initial global state, such that state transitions occur according to $A$. We use the notion of *time* $t$ *d*uring a run $r$ to refer to the $t^{th}$ action in $r$ and the global state that ensues it. A *run fragment* is a contiguous subsequence of a run. An operation invoked before time $t$ in run

$r$ is *complete* at time $t$ if its response event occurs before time $t$ in $r$; otherwise it is *pending* at time $t$. We assume that runs are *well-formed* [16], in that each process's first action is an invocation of some operation, and a process does not invoke an operation before receiving a response to its last invoked one.

We say that operation $op_i$ *precedes* operation $op_j$ in a run $r$, if $op_i$'s response occurs before $op_j$'s invocation in $r$. Operations $op_i$ and $op_j$ are *concurrent* in run $r$, if $op_i$ does not precede $op_j$ and $op_j$ does not precede $op_i$ in $r$. A *sequential run* is one with no concurrent operations. Two runs are *equivalent* if every process performs the same sequence of operations (with the same return values) in both, where operations that are pending in one can either be included in or excluded from the other.

### 2.2 Dynamic Storage

The distributed storage service we consider is a *dynamic multi-writer, multi reader (MWMR) register* [2,13,15,18,23,26], which stores a value $v$ from a domain $\mathbb{V}$, and offers an interface for invoking *read*, *write*, and *reconfig* operations. Initially, the register holds some initial value $v_0 \in \mathbb{V}$. A *read* operation takes no parameters and returns a value from $\mathbb{V}$, and a *write* operation takes a value from $\mathbb{V}$ and returns "ok". We define *Changes* to be the set $\{remove, add\} \times \Pi$, and call any subset of Changes a *set of changes*. For example, $\{\langle add, p_3\rangle, \langle remove, p_2\rangle\}$ is a set of changes. A *reconfig* operation takes as a parameter a set of changes and returns "ok". For simplicity, we assume that a process that has been removed is not added again.

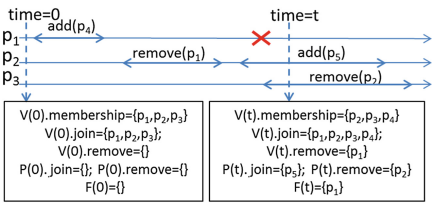



**Fig. 1.** Notation illustration. $add(p)$ ($remove(p)$) represents $reconfig(\langle add, p\rangle)$ (respectively, $reconfig(\langle remove, p\rangle)$).

*Notation.* For every subset $w$ of *Changes*, the *removal set* of $w$, denoted $w.remove$, is $\{p_i | \langle remove, p_i\rangle \in w\}$; the *join set* of $w$, denoted $w.join$, is $\{p_i | \langle add, p_i\rangle \in w\}$; and the *membership* of $w$, denoted $w.membership$, is $w.join \setminus w.remove$. For example, for a set $w = \{\langle add, p_1\rangle, \langle remove, p_1\rangle, \langle add, p_2\rangle\}$, $w.join = \{p_1, p_2\}$, $w.remove = \{p_1\}$, and $w.membership = \{p_2\}$. For a time $t$ in a run $r$, we denote by $V(t)$ the union of all sets $q$ s.t. *reconfig(q)* completes before time $t$ in $r$. A *configuration* is a finite set of processes, and the *current configuration at time* $t$ is $V(t).membership$. We assume that only processes in $V(t).membership$ invoke operations at time $t$. The initial set of processes $\Pi_0 \subset \Pi$ is known to all and we say, by convention, that $reconfig(\{\langle add, p\rangle | p \in \Pi_0\})$ completes at time 0, i.e., $V(0).membership = \Pi_0$.

We define $P(t)$ to be the set of *pending changes* at time $t$ in run $r$, i.e., the set of all changes included in pending reconfig operations. We denote by $F(t)$ the set of processes that have failed before time $t$ in $r$; initially, $F(0) = \{\}$. For

a series of arbitrary sets $S(t)$, $t \in \mathbb{N}$, we define $S(*) \triangleq \bigcup_{t \in \mathbb{N}} S(t)$. The notation is illustrated in Fig. 1.

*Correct processes and fairness.* A process $p$ is *correct* if $p \in V(*).join \setminus F(*)$. A run $r$ is *fair* if every send action by a correct process that is enabled infinitely often eventually occurs, and every message sent by a correct process $p_i$ to a correct process $p_j$ is eventually received at $p_j$. Note that messages sent to a faulty process from a correct one may or may not be received. A process $p$ is *active* if $p$ is correct, and $p \notin P(*).remove$.

*Service specification.* A *linearization* of a run $r$ is an equivalent sequential run that preserves $r$'s operation precedence relation and the service's sequential specification. The sequential specification for a register is as follows: A read returns the latest written value, or $v_0$ if none was written. An MWMR register is *atomic*, also called *linearizable* [16], if every run has a linearization. Lamport [20] defines a *safe* single-writer register. Here, we generalize the definition to multi-writer registers in a weak way in order to strengthen the impossibility result. Intuitively, if a read is not concurrent with any write we require it to return a value that reflects some possible outcome of the writes that precede it; otherwise we allow it to return an arbitrary value. Formally: An MWMR register is *safe* if for every run $r$ for every *read* operation $rd$ that has no concurrent *writes* in $r$, there is a linearization of the subsequence of $r$ consisting of $rd$ and the *write* operations in $r$.

A *wait-free* service guarantees that every active process's operation completes regardless of the actions of other processes.

*Failure model and reconfiguration.* The reconfig operations determine which processes are allowed to fail at any given time. Static storage algorithms [3] tolerate failures of a minority of their (static) universe. At a time $t$ when no reconfig operations are ongoing, the dynamic failure condition may be simply defined to allow less than $|V(t)membership|/2$ failures of processes in $V(t).membership$. When there are pending additions and removals, the rule must be generalized to take them into account. For our algorithm in Sect. 4, we adopt a generalization presented in previous works [1,2,18,26]:

**Definition 1 (minority failures).** *A model allows* minority failures *if at all times $t$ in $r$, fewer than $|V(t).membership \setminus P(t).remove|/2$ processes out of $V(t).membership \cup P(t).join$ are in $F(t)$.*

Note that this failure condition allows processes whose remove operations have completed to be (immediately) safely switched off as it only restricts failures out of the current membership and pending joins. We say that a service is *reconfigurable* if failures of processes in $V(t).remove$ are unrestricted.

In order to strengthen our lower bound in Sect. 3 we weaken the failure model. Like FLP [12], our lower bound applies as long as at least *one* process can fail. Formally, a failure is allowed whenever all failed processes have been removed and

the current membership consists of at least three processes[1]. We call such a state "clean", captured by the following predicate: $clean(t) \triangleq (V(t).membership \cup P(t).join) \cap F(t) = \{\} \wedge |V(t).membership \setminus P(t).remove| \geq 3$. The minimal failure condition is thus defined as follows:

**Definition 2 (minimal failure).** *A model allows* minimal failure *if in every run $r$ ending at time $t$ when $clean(t)$, for every process $p \in V(t).membership \cup P(t)$, there is an extension of $r$ where $p$ fails at time $t+1$.*

Notice that the minority failure condition allows minimal failure, and so all algorithms that assume minority failures [1,2,18,26] are a fortiori subject to our lower bound, which is proven for minimal failures.

## 3 Impossibility of Wait-Free Dynamic Safe Storage

In this section we prove that there is no implementation of wait-free dynamic safe storage in a model that allows minimal failures. We construct a fair run with infinitely many reconfiguration operations in which a slow process $p$ never completes its write operation. We do so by delaying all of $p$'s messages. A message from $p$ to a process $p_i$ is delayed until $p_i$ is removed, and we make sure that all processes except $p$ are eventually removed and replaced.

**Theorem 1.** *There is no algorithm that emulates wait-free dynamic safe storage in an asynchronous system allowing minimal failures.*

*Proof (Proof (Theorem 1)).* Assume by contradiction that such an algorithm $A$ exists. We prove two lemmas about $A$.

**Lemma 1.** *Consider a run $r$ of $A$ ending at time $t$ s.t. $clean(t)$, and two processes $p_i, p_j \in V(t).membership$. Extend $r$ by having $p_j$ invoke operation $op$ at time $t+1$. Then there exists an extension of $r$ where (1) $op$ completes at some time $t' > t$,(2) no process receives a message from $p_i$ between $t$ and $t'$, and (3) no process fails and no operations are invoked between $t$ and $t'$.*

*Proof (Lemma 1).* By the minimal failure condition, $p_i$ can fail at time $t+2$. Consider a fair extension $\sigma_1$ of $r$, in which $p_i$ fails at time $t+2$ and all of its in-transit messages are lost, no other process fails, and no operations are invoked. By wait-freedom, $op$ eventually completes at some time $t_1$ in $\sigma_1$. Since $p_i$ fails and all its outstanding messages are lost, then from time $t$ to $t_1$ in $\sigma_1$ no process receives any messages from $p_i$. Now let $\sigma_2$ be identical to $\sigma_1$ except that $p_i$ does not fail, but all of its messages are delayed. Note that $\sigma_1$ and $\sigma_2$ are indistinguishable to all processes except $p_i$. Thus, $op$ returns at time $t_1$ also in $\sigma_2$.

[1] Note that with fewer than three processes, even static systems cannot tolerate failures [3].

**Lemma 2.** *Consider a run $r$ of $A$ ending at time $t$ s.t $clean(t)$. Let $v_1 \in \mathbb{V} \setminus \{v_0\}$ be a value s.t. no process invokes $write(v_1)$ in $r$. If we extend $r$ fairly so that $p_i$ invokes $w = write(v_1)$ at time $t+1$ which completes at some time $t_1 > t+1$ s.t. $clean(t')$ for all $t < t' \leq t_1$ then in the run fragment between $t+1$ and $t_1$, some process $p_k \neq p_i$ receives a message sent by $p_i$.*

*Proof (Lemma 2).* Assume by way of contradiction that in the run fragment between $t+1$ and $t_1$ no process $p_k \neq p_i$ receives a message sent by $p_i$, and consider a run $r'$ that is identical to $r$ until time $t_1$ except that $p_i$ does not invoke $w$ at time $t$. Now assume that some process $p_j \neq p_i$ invokes a *read* operation $rd$ at time $t_1 + 1$ in $r'$. By the assumption, $clean(t_1)$ and therefore $clean(t_1 + 1)$. Thus, by Lemma 1, there is a run fragment $\sigma$ beginning at the final state of $r'$ (time $t_1 + 1$), where $rd$ completes at some time $t_2$, s.t. between $t_1 + 1$ and $t_2$ no process receives a message from $p_i$. Since no process invokes $write(v_1)$ in $r'$, and no writes are concurrent with the read, by safety, $rd$ returns some $v_2 \neq v_1$.

Now notice that all global states from time $t$ to time $t_1$ in $r$ and $r'$ are indistinguishable to all processes except $p_i$. Thus, we can continue run $r$ with an invocation of read operation $rd'$ by $p_j$ at time $t_1$, and append $\sigma$ to it. Operation $rd'$ hence completes and returns $v_2$. A contradiction to safety.

To prove the theorem, we construct an infinite fair run $r$ in which a *write* operation of an active process never completes, in contradiction to wait-freedom.

Consider some initial global state $c_0$, s.t. $P(0) = F(0) = \{\}$ and $V(0).membership = \{p_1 \dots p_n\}$, where $n \geq 3$. An illustration of the run for $n = 4$ is presented in Fig. 2. Now, let process $p_1$ invoke a write operation $w$ at time $t_1 = 0$, and do the following:

Let process $p_n$ invoke *reconfig(q)* where $q = \{\langle add, p_j \rangle | n + 1 \leq j \leq 2n - 2\}$ at time $t_1$. The state at the end of $r$ is clean (i.e., $clean(t_1)$). So by Lemma 1, we can extend $r$ with a run fragment $\sigma_1$ ending at some time $t_2$ when *reconfig(q)* completes, where no process $p_j \neq p_1$ receives a message from $p_1$ in $\sigma_1$, no other operations are invoked, and no process fails.

Then, at time $t_2 + 1$, $p_n$ invokes *reconfig*$(q')$, where $q' = \{\langle remove, p_j \rangle | 2 \leq j \leq n - 1\}$. Again, the state is clean and thus by Lemma 1 again, we can extend $r$ with a run fragment $\sigma_2$ ending at some time $t_3$ when *reconfig(q')* completes s.t. no process $p_j \neq p_1$ receives a message from $p_1$ in $\sigma_2$, no other operations are invoked, and no process fails.

Recall that the minimal failures condition satisfies reconfigurability, i.e., all the processes in $V(t_3).remove$ can be in $F(t_3)$ (fail). Let the processes in $\{p_j \mid 2 \leq j \leq n - 1\}$ fail at time $t_3$, and notice that the fairness condition does not mandate that they receive messages from $p_1$. Next, allow $p_1$ to perform all its enabled actions till some time $t_4$.

Now notice that at $t_4$, $|V(t_4).membership| = n$, $P(t_4) = \{\}$, $(V(t_4).membership \cup P(t_4).join) \cap F(t_4) = \{\}$, and $|V(t_4).membership \setminus P(t_4).removal| \geq 3$. We can rename the processes in $V(t_4).membership$ (except $p_1$) so that the process that performed the remove and add operations becomes

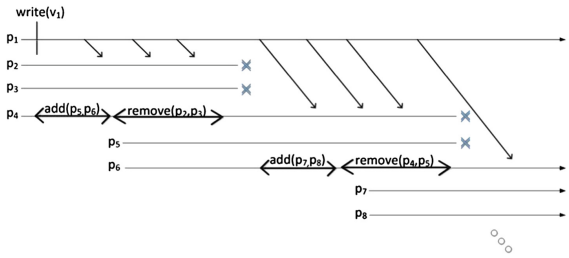



**Fig. 2.** Illustration of the infinite run for $n = 4$.

$p_2$, and all others get names in the range $p_3 \dots p_n$. We can then repeat the construction above. By doing so infinitely many times, we get an infinite run $r$ in which $p_1$ is active and no process ever receives a message from $p_1$. However, all of $p_1$'s enabled actions eventually occur. Since no process except $p_1$ is correct in $r$, the run is fair. In addition, since $clean(t)$ for all $t$ in $r$, by the contrapositive of Lemma 2, $w$ does not complete in $r$, and we get a violation of wait-freedom.

## 4 Oracle-Based Dynamic Atomic Storage

We present an algorithm that circumvents the impossibility result of Sect. 3 using a failure detector. In this section we assume the minority failure condition. In Sect. 4.1, we define a dynamic eventually perfect failure detector. In Sect. 4.2, we describe an algorithm, based on dynamic state machine replication, that uses the failure detector to implement a wait-free dynamic atomic MWMR register. The algorithm's correctness is proven in Appendix A.

### 4.1 Dynamic Failure Detector

Since the set of processes is potentially infinite, we cannot have the failure detector report the status of all processes as static failure detectors typically do. Dynamic failure detectors addressing this issue have been defined in previous works, either providing a set of processes that have been excluded from or included into the group [22], or assuming that there is eventually a fixed set of participating processes [10]. In our model, we do not assume that there is eventually a fixed set of participating processes, as the number of reconfig operations can be infinite. And we do not want the failure detector to answer with a list of processes, because in dynamic systems, this gives additional information about participating processes that could have been unknown to the inquiring process, and thus it is not clear how such a failure detector can be implemented.

Instead, our dynamic failure detector is queried separately about each process. For each query, it answers either *fail* or *ok*. It can be wrong for an unbounded period, but for each process, it eventually returns a correct answer. Formally, a *dynamic eventually perfect* failure detector, $\Diamond P^D$, satisfies two properties:

- **Strong completeness:** For each process $p_i$ that fails at time $t_i$, there is a time $t > t_i$ s.t. the failure detector answers *fail* to every query about $p_i$ after time $t$.
- **Eventual strong accuracy:** There exists a time $t$, called the *stabilization time*, s.t. the failure detector answers *ok* to every query at any time $t' > t$ about a correct process in $V(t').join$.

Note that $\Diamond P^D$ can be implemented in a standard way in the eventually (partially) synchronous model by pinging the queried process and waiting for a response until a timeout.

### 4.2 Dynamic Storage Algorithm

We first give the overview of our algorithm and and then present the full description.

**Algorithm overview.** The key to achieving liveness with unbounded reconfig operations is a novel helping mechanism, which is based on our failure detector. Intuitively, the idea is that every process tries to help all other processes it believes are correct, (according to its failure detector), to complete their concurrent operations together with its own. At the beginning of an operation, a process $p$ queries all other processes it knows about for the operations they currently perform. The failure detector is needed in order to make sure that (1) $p$ does not wait forever for a reply from a faulty process (achieved by strong completeness), and (2) every slow correct process eventually gets help (achieved by eventual strong accuracy).

*State machine emulation of a register.* We use a state machine *sm* to emulate a wait-free atomic dynamic register, *DynaReg*. Every process has a local replica of *sm*, and we use consensus to agree on *sm*'s state transitions. Notice that each process is equipped with a failure detector FD of class $\Diamond P^D$, so consensus is solvable under the assumption of a correct majority in a given configuration [21].

Each instance of consensus runs in some static configuration $c$ and is associated with a unique timestamp. A process participates in a consensus instance by invoking a *propose* operation with the appropriate configuration and timestamp, as well as its proposed decision value. Consensus then responds with a *decide* event, so that the following properties are satisfied: *Uniform Agreement* – every two decisions are the same. *Validity* – every decision was previously proposed by one of the processes in $c$. *Termination* – if a majority of $c$ is correct, then eventually every correct process in $c$ decides. We further assume that a consensus instance does not decide until a majority of the members of the configuration propose in it.

The *sm* (lines 2–5 in Algorithm 1) keeps track of dynaReg's value in a variable *val*, and the configuration in a variable *cng*, containing both a list of processes, *cng.mem*, and a set of removed processes, *cng.rem*. Write operations change

*val*, and reconfig operations change *cng*. A consensus decision may bundle a number of operations to execute as a single state transition of *sm*. The number of state transitions executed by *sm* is stored in the variable *ts*. Finally, the array *lastOps* maps every process $p$ in *cng.mem* to the sequence number (based on $p$'s local count) of $p$'s last operation that was performed on the emulated DynaReg together with its result.

Each process partakes in at most one consensus at a time; this consensus is associated with timestamp *sm.ts* and runs in *sm.cng.mem*. In every consensus, up to $|sm.cng.mem|$ ordered operations on the emulated DynaReg are agreed upon, and *sm*'s state changes according to the agreed operations. A process's *sm* may change either when consensus decides or when the process receives a newer *sm* from another process, in which case it skips forward. So *sm* goes through the same states in all the processes, except when skipping forward. Thus, for every two processes $p_k, p_l$, if $sm_k.ts = sm_l.ts$, then $sm_k = sm_l$. (A subscript $i$ indicates the variable is of process $p_i$.)

*Helping.* The problematic scenario in the impossibility proof of Sect. 3 occurs because of endless reconfig operations, where a slow process is never able to communicate with members of its configuration before they are removed. In order to circumvent this problem, we use FD to implement a helping mechanism. When proposing an operation, process $p_i$ tries to help other processes in two ways: first, it helps them complete operations they may have successfully proposed in previous rounds (consensuses) but have not learned about their outcomes; and second, it proposes their new operations. To achieve the first, it sends a helping request with its *sm* to all other processes in $sm_i.cng.mem$. For the second, it waits for each process to reply with a help reply containing its latest invoked operation, and then proposes all the operations together. Processes may fail or be removed, so $p_i$ cannot wait for answers forever. To this end, we use FD. For every process in $sm_i.cng.mem$ that has not been removed, $p_i$ repeatedly inquires FD and waits either for a reply from the process or for an answer from FD that the process has failed. Notice that the strong completeness property guarantees that $p_i$ will eventually continue, and strong accuracy guarantees that every slow active process will eventually receive help in case of endless reconfig operations.

Nevertheless, if the number of reconfig operations is finite, it may be the case that some slow process is not familiar with any of the correct members in the current configuration, and no other process performs an operation (hence, no process is helping). To ensure progress in such cases, every correct process periodically sends its *sm* to all processes in its *sm.cng.mem*.

*State survival.* Before the reconfig operation can complete, the new *sm* needs to propagate to a majority of the new configuration, in order to ensure its survival. Therefore, after executing the state transition, $p_i$ sends $sm_i$ to $sm_i.cng$ members and waits until it either receives acknowledgements from a majority or learns of a newer *sm*. Notice that in the latter case, consensus in $sm_i.cng.mem$ has decided, meaning that at least a majority of $sm_i.cng.mem$ has participated in it, and so have learned of it.

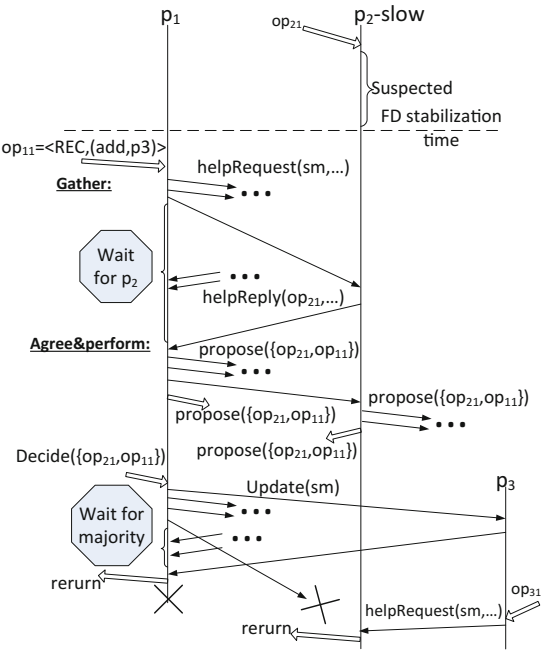



**Fig. 3.** Flow illustration: process $p_2$ is slow. After stabilization time, process $p_1$ helps it by proposing its operation. Once $p_2$'s operation is decided, it is reflected in every up-to-date *sm*. Therefore, even if $p_1$ fails before informing $p_2$, $p_2$ receives from the next process that performs an operation, namely $p_3$, an *sm* that reflects its operation, and thus returns. Line arrows represent messages, and block arrows represent operation or consensus invocations and responses.

*Flow example.* The algorithm flow is illustrated in Fig. 3. In this example, a slow process $p_2$ invokes operation $op_{21}$ before FD's stabilization time, $ST$. Process $p_1$ invokes operation $op_{11} = \langle add, p_3 \rangle$ after ST. It first sends *helpRequest* to $p_2$ and waits for it to reply with *helpReply*. Then it proposes $op_{21}$ and $op_{11}$ in a consensus. When *decide* occurs, $p_1$ updates its *sm*, sends it to all processes, and waits for majority. Then $op_{11}$ returns and $p_1$ fails before $p_2$ receives its update message. Next, $p_3$ invokes a *reconfig* operation, but this time when $p_2$ receives *helpRequest* with the up-to-date *sm* from $p_3$, it notices that its operation has been performed, and $op_{21}$ returns.

**Detailed description.** The data structure of process $p_i$ is given in Algorithm 1. The type *Ops* defines the representation of operations. The emulated state machine, $sm_i$, is described above. Integer $opNum_i$ holds the sequence number of $p_i$'s current operation; $ops_i$ is a set that contains operations that need to be completed for helping; the flag $pend_i$ is a boolean that indicates whether or not $p_i$ is participating in an ongoing consensus; and $myOp_i$ is the latest operation invoked at $p_i$.

**Algorithm 1.** Data structure of process $p_i$

1: $Ops \triangleq \{\langle RD, \perp\rangle\} \cup \{\langle WR, v\rangle \mid v \in \mathbb{V}\} \cup \{\langle REC, c\rangle \mid c \subset Changes\}$
2: $sm_i.ts \in \mathbb{N}$, initially 0
3: $sm_i.value \in \mathbb{V}$, initially $v_0$
4: $sm_i.cng = \langle mem, rem\rangle$, where $mem, rem \subset \Pi$, initially $\langle \Pi_0, \{\}\rangle$
5: $sm_i.lastOps$ is a vector of size $|sm_i.cng.mem|$, where $\forall p_j \in sm_i.cng.mem$, $sm_i.lastOps[j] = \langle num, res\rangle$,
where $num \in \mathbb{N}$, $res \in \mathbb{V} \cup \{$"ok"$\}$, initially $\langle 0,$ "ok" $\rangle$
6: $pend_i \in \{true, false\}$, initially $false$
7: $opNum_i \in \mathbb{N}$, initially 0
8: $ops_i \subset \Pi \times Ops \times \mathbb{N}$ , initially $\{\}$
9: $myOp_i \in operation$, initially $\perp$

The algorithm of process $p_i$ is presented in Algorithms 2 and 3. We execute every event handler, (operation invocation, message receiving, and consensus decision), atomically excluding wait instructions; that is, other event handlers may run after the handler completes or during a wait (lines 16,18,27 in Algorithm 2). The algorithm runs in two phases. The first, *gather*, is described in Algorithm 2 lines 11–16 and in Algorithm 3 lines 52–58. Process $p_i$ first increases its operation number $opNum_i$, writes $op$ together with $opNum_i$ to the set of operations $ops_i$, and sets $myOp_i$ to be $op$. Then it sends $\langle$"helpRequest", $\ldots\rangle$ to every member of $A = sm_i.cng.mem$ (line 15), and waits for each process in $A$ that is not suspected by the FD or removed to reply with $\langle$"helpReply", $\ldots\rangle$. Notice that $sm_i$ may change during the wait because messages are handled, and $p_i$ may learn of processes that have been removed.

When $\langle$"helpRequest", $num, sm\rangle$ is received by process $p_j \neq p_i$, if the received $sm$ is newer than $sm_j$, then process $p_j$ adopts $sm$ and abandons any previous consensus. Either way, $p_j$ sends $\langle$"helpReply", $\ldots\rangle$ with its current operation $myOp_j$ in return.

Upon receiving $\langle$"helpReply", $opNum_i, op, num\rangle$ that corresponds to the current operation number $opNum_i$, process $p_i$ adds the received operation $op$, its number $num$, and the identity of the sender to the set $ops_i$.

At the end of this phase, process $p_i$ holds a set of operations, including its own, that it tries to agree on in the second phase (the order among this set is chosen deterministically, as explained below). Note that $p_i$ can participate in at most one consensus per timestamp, and its propose might end up not being the decided one, in which case it may need to propose the same operations again. Process $p_i$ completes $op$ when it discovers that $op$ has been performed in $sm_i$, whether by itself or by another process.

The second phase appears in Algorithm 2 lines 17–28, and in Algorithm 3 lines 31–51. In line 17, $p_i$ checks if its operation has not been completed yet. In line 18, it waits until it does not participate in any ongoing consensus ($pend_i = false$) or some other process helps it complete $op$. Recall that during a wait, other events can be handled. So if a message with an up-to-date $sm$ is received during the wait, $p_i$ adopts the $sm$. In case $op$ has been completed in $sm$, $p_i$ exits

**Algorithm 2.** Process $p_i$'s algorithm: performing operations

```
10: upon invoke operation(op) do
                                                          ▷ phase 1: gather
11:     opNum_i ← opNum_i + 1
12:     ops_i ← {⟨p_i, op, opNum_i⟩}
13:     myOp_i ← op
14:     A ← sm_i.cng.mem
15:     for all p ∈ A send ⟨"helpRequest", opNum_i, sm_i⟩ to p
16:     for all p ∈ A wait for ⟨"helpReply", opNum_i, ...⟩ from p or p is suspected or p ∈ sm_i.cng.rem
                                                          ▷ phase 2: agree&perform
17:     while sm_i.lastOps[i].num ≠ opNum_i
18:         wait until ¬pend_i or sm_i.lastOps[i].num = opNum_i
19:         if sm_i.lastOps[i].num = opNum_i then break
20:         pend_i ← true
21:         Req ← {⟨p_j, op, num⟩ ∈ ops_i | num > sm_i.lastOps[j].num}
22:         propose(sm_i.cng, sm_i.ts, Req)
23:         for all p ∈ sm_i.cng.mem send ⟨"propose", sm_i, Req⟩ to p
24:     if op.type = REC
25:         ts ← sm_i.ts
26:         for all p ∈ sm_i.cng.mem send ⟨"update", sm_i, opNum_i⟩ to p
27:         wait for ⟨"ACK", opNum_i⟩ from majority of sm_i.cng.mem or sm_i.ts > ts
28:     return sm_i.lastOps[i].res

29: periodically:
30:     for all p ∈ sm_i.cng.mem send ⟨"update", sm_i, ⊥⟩ to p
```

the main while (line 19). Otherwise, $p_i$ waits until it does not participate in any ongoing consensus. This can be the case if (1) $p_i$ has not proposed yet, (2) a message with a newer *sm* was received and a previous consensus was subsequently abandoned, or (3) a *decide* event has been handled. In all cases, $p_i$ marks that it now participates in consensus in line 20, prepares a new request *Req* with the operations in $ops_i$ that have not been performed yet in $sm_i$ in line 27, proposes *Req* in the consensus associated with $sm_i.ts$, and sends $\langle$"propose", ...$\rangle$ to all the members of $sm_i.cng.mem$.

When $\langle$"propose", $sm, Req \ldots\rangle$ is received by process $p_j \neq p_i$, if the received *sm* is more updated than $sm_j$, then process $p_j$ adopts *sm*, abandons any previous consensus, proposes *Req* in the consensus associated with $sm.ts$, and forwards the message to all other members of $sm_j.cng.mem$. The same is done if *sm* is identical to $sm_j$ and $p_j$ has not proposed yet in the consensus associated with $sm_j.ts$. Otherwise, $p_j$ ignores the message.

The event $decide_i(sm.cng, sm_i.ts, Req)$ indicates a decision in the consensus associated with $sm_i.ts$. When this occurs, $p_i$ performs all the operations in *Req* and changes $sm_i$'s state. It sets the value of the emulated DynaReg, $sm_i.value$, to be the value of the *write* operation of the process with the lowest id, and updates $sm_i.cng$ according to the *reconfig* operations. In addition, for every $\langle p_j, op, num\rangle \in Req$, $p_i$ writes to $sm_i.lastOps[j]$, *num* and *op*'s response, which is "ok" in case of a *write* or a *reconfig*, and $sm_i.value$ in case of a *read*. Next, $p_i$ increases $sm_i.ts$ and sets $pend_i$ to false, indicating that it no longer participates in any ongoing consensus.

Finally, after *op* is performed, $p_i$ exits the main while. If *op* is not a *reconfig* operation, then $p_i$ returns the result, which is stored in $sm_i.lastOps[i].res$. Otherwise, before returning, $p_i$ has to be sure that a majority of $sm_i.cng.mem$

receives $sm_i$. It sends $\langle$"update", $sm, \ldots\rangle$ to all the processes in $sm_i.cng.mem$ and waits for $\langle$"ACK", $\ldots\rangle$ from a majority of them. Notice that it may be the case that there is no such correct majority due to later reconfig operations and failures, so, $p_i$ stops waiting when a more updated $sm$ is received, which implies that a majority of $sm_i.cng.mem$ has already received $sm_i$ (since a majority is needed in order to solve consensus).

Upon receiving $\langle$"update", $sm, num\rangle$ with a new $sm$ from process $p_i$, process $p_j$ adopts $sm$ and abandons any previous consensus. In addition, if $num \neq \perp$, $p_j$ sends $\langle$"ACK", $num\rangle$ to $p_i$ (Algorithm 3 lines 59–63).

Beyond handling operations, in order to ensure progress in case no operations are invoked from some point on, every correct process periodically sends $\langle$"update", $sm, \perp\rangle$ to all processes in its $sm.cng.mem$ (Algorithm 2 line 30).

**Algorithm 3.** Process $p_i$'s algorithm: event handlers

```
31: upon decide_i(sm_i.cng, sm_i.ts, Req) do
32:     W ← {⟨p, value⟩ | ⟨p, ⟨WR, value⟩, num⟩ ∈ Req}
33:     if W ≠ {}                                   ▷ deterministically choose one of the writes to be the last
34:         sm_i.value ← value with smallest p in W
35:     for all ⟨p_j, op, num⟩ ∈ Req                                              ▷ apply op to sm
36:         if op.type = WR
37:             sm_i.lastOps[j] ← ⟨num, "ok"⟩
38:         else if op.type = RD
39:             sm_i.lastOps[j] ← ⟨num, sm_i.value⟩
40:         else
41:             sm_i.cng.rem ← sm_i.cng.rem ∪ {p | ⟨remove, p⟩ ∈ op.changes}
42:             sm_i.cng.mem ← sm_i.cng.mem ∪ {p | ⟨add, p⟩ ∈ op.changes} \ sm_i.cng.rem
43:             sm_i.lastOps[j] ← ⟨num, "ok"⟩
44:     sm_i.ts ← sm_i.ts + 1
45:     pend_i ← false

46: upon receiving ⟨"propose", sm, Req⟩ from p_j do
47:     if (sm_i.ts > sm.ts) or (sm_i.ts = sm.ts ∧ pend_i = true) then return
48:     sm_i ← sm
49:     pend_i ← true
50:     propose(sm_i.cng, sm_i.ts, Req)
51:     for all p ∈ sm_i.cng.mem send ⟨"propose", sm_i, Req⟩ to p

52: upon receiving ⟨"helpRequest", num, sm⟩ from p_j do
53:     if sm_i.ts < sm.ts then                                                   ▷ learn new sm
54:         sm_i ← sm
55:         pend_i ← false
56:     send ⟨"helpReply", num, myOp_i, opNum_i⟩

57: upon receiving ⟨"helpReply", opNum_i, op, num⟩ from p_j do
58:     ops_i ← ops_i ∪ ⟨p_j, op, num⟩

59: upon receiving ⟨"update", sm, num⟩ from p_j do
60:     if sm_i.ts < sm.ts then                                                   ▷ learn new sm
61:         sm_i ← sm
62:         pend_i ← false
63:     if num ≠⊥ then send ⟨"ACK", num⟩ to p_j
```

## 5 Conclusion

We proved that in an asynchronous API-based reconfigurable model allowing at least one failure, without restricting the number of reconfigurations, there is no way to emulate dynamic safe wait-free storage. We further showed how to circumvent this result using a dynamic eventually perfect failure detector: we presented an algorithm that uses such a failure detector in order to emulate a wait-free dynamic atomic MWMR register.

Our dynamic failure detector is (1) sufficient for this problem, and (2) can be implemented in a dynamic eventually synchronous [11] setting with no restriction on reconfiguration rate. An interesting question is whether a weaker such failure detector exists. Note that when the reconfiguration rate is bounded, dynamic storage is attainable without consensus, thus such a failure detector does not necessarily have to be strong enough for consensus.

## A Correctness Proof

In Sect. A.1 we prove that our algorithm satisfies atomicity, and in Sect. A.2 wait-freedom.

### A.1 Atomicity

Every operation is uniquely defined by the process that invoked it and its local number. During the proof we refer to operation *op* invoked by process $p_i$ with local number $opNum_i = n$ as the tuple $\langle p_i, op, n\rangle$. We begin the proof with three lemmas that link completed operations to *sm* states.

**Lemma 3.** *Consider operation op invoked by some process $p_i$ in r with local number $opNum_i = n$. If op returns in r at time t, then there is at least one request* Req *that contains $\langle p_i, op, n\rangle$ and has been chosen in a consensus in r before time t.*

*Proof.* When operation *op* return, $sm_i.lastOps[i].num = n$ (line 17 or 18 in Algorithm 2). Processes update *sm* during a decide handler, or when a newer *sm* is received, and the first update occurs when some process $p_j$ writes $n$ to $sm_j.lastOps[i].num$ during a decide handler. In the decide handler, $n$ is written to $sm.lastOps[i].num$ when the chosen request in the corresponding consensus contains $\langle p_i, op, n\rangle$.

**Lemma 4.** *For two processes $p_i, p_j$, let t be a time in a run r in which neither $p_i$ or $p_j$ is executing a decide handler. Then at time t, if $sm_i.ts = sm_j.ts$, then $sm_i = sm_j$.*

*Proof.* We prove by induction on timestamps. Initially, all correct processes have the same *sm* with timestamp 0. Now consider timestamp $TS$, and assume that for every two processes $p_i, p_j$ at any time not during the execution of decide

handlers, if $sm_i.ts = sm_j.ts = TS$, then $sm_i = sm_j$. Processes increase their $sm.ts$ to $TS+1$ either at the end of a *decide* handler associated with $TS$ or when they receive a message with $sm$ s.t. $sm.ts = TS+1$. By the agreement property of consensus and by the determinism of the algorithm, all the processes that perform the *decide* handler associated with $TS$ perform the same operations, and therefore move $sm$ (at the end of the handler) to the same state. It is easy to show by induction that all the processes that receive a message with $sm$ s.t. $sm.ts = TS + 1$ receive the same $sm$. The lemma follows.

**Observation 1.** *For two process $p_i, p_j$, let $sm_1$ and $sm_2$ be the values of $sm_j$ at two different times in a run $r$. If $sm_1.ts \geq sm_2.ts$, then $sm_1.lastOps[i].num \geq sm_2.lastOps[i].num$.*

**Lemma 5.** *Consider operation $\langle p_i, op, opNum_i \rangle$ invoked in $r$ with $opNum_i = n$. Then $\langle p_i, op, n \rangle$ is part of at most one request that is chosen in a consensus in $r$.*

*Proof.* Assume by way of contradiction that $\langle p_i, op, n \rangle$ is part of more than one request that is chosen in a consensus in $r$. Now consider the earliest one, $Req$, and assume that it is chosen in a consensus associated with timestamp $TS$. At the end of the *decide* handler associated with timestamp $TS$, $sm.lastOps[i].num = n$ and the timestamp is increased to $TS+1$. Thus, by Lemma 4 $sm.lastOps[i].num = n$ holds for every $sm$ s.t. $sm.ts = TS + 1$. Consider now the next request, $Req_1$, that contains $\langle p_i, op, n \rangle$, and is chosen in a consensus. Assume that this consensus associated with timestamp $TS'$, and notice that $TS' > TS$. By the validity of consensus, this request is proposed by some process $p_j$, when $sm_j.ts$ is equal to $TS'$. By Observation 1, at this point $sm_j.lastOps[i].num \geq n$, and therefore $p_j$ does not include $\langle p_i, op, n \rangle$ in $Req_1$ (line 27 in Algorithm 2). A contradiction.

Based on the above lemmas, we can define, for each run $r$, a linearization $\sigma_r$, where operations are ordered as they are chosen for execution on $sm$'s in $r$.

**Definition 3.** *For a run $r$, we define the sequential run $\sigma_r$ to be the sequence of operations decided in consensus instances in $r$, ordered by the order of the chosen requests they are part of in $r$. The order among operations that are part of the same chosen request is the following: first all* writes*, then all* reads*, and finally, all* reconfig *operations. Among each type, operations are ordered by the process ids of the processes that invoked them, from the highest to the lowest.*

Note that for every run $r$, the sequential run $\sigma_r$ is well defined. Moreover, $\sigma_r$ contains every completed operation in $r$ exactly once, and every invoked operation at most once.

In order to prove atomicity we show that (1) $\sigma_r$ preserves $r$'s real time order (Lemma 6); and (2) every *read* operation $rd$ in $r$ returns the value that was written by the last *write* operation that precedes $rd$ in $\sigma_r$, or $\perp$ if there is no such operation (Lemma 7).

**Lemma 6.** *If operation $op_1$ returns before operation $op_2$ is invoked in $r$, then $op_1$ appears before $op_2$ in $\sigma_r$.*

*Proof.* By Lemma 3, $op_1$ is part of a request $Req_1$ that is chosen in a consensus before $op_2$ is invoked, and thus $op_2$ cannot be part of $Req_1$ or any other request that is chosen before $Req_1$. Hence $op_1$ appears before $op_2$ in $\sigma_r$.

**Lemma 7.** *Consider read operation $rd = \langle p_i, RD, n\rangle$ in $r$, which returns a value $v$. Then $v$ is written by the last write operation that precedes $rd$ in $\sigma_r$, or $v = \perp$ if there is no such operation.*

*Proof.* By Lemmas 3 and 5, $rd$ is part of exactly one request $Req_1$ that is chosen in a consensus, associated with some timestamp $TS$. Thus $sm.lastOps[i]$ is set to $\langle n, val\rangle$ in the *decide* handler associated with $TS$. By Lemma 4, $sm.lastOps[i] = \langle n, val\rangle$ for all $sm$ s.t. $sm.ts = TS + 1$. By Lemma 5 and since we consider only well-formed runs, $sm_i.lastOps[i] = \langle n, val\rangle$ when $rd$ returns, and therefore $rd$ returns $val$. Now consider three cases:

- There is no *write* operation in $Req_1$ or in any request that was chosen before $Req_1$ in $r$. In this case, there is no *write* operation before $rd$ in $\sigma_r$, and no process writes to $sm.value$ before $sm.lastOps[i]$ is set to $\langle n, val\rangle$, and therefore, $rd$ returns $\perp$ as expected.
- There is a *write* operation in $Req_1$ in $r$. Consider the *write* operation $w$ in $Req_1$ that is invoked by the process with the lowest id, and assume its argument is $v'$. Notice that $w$ is the last *write* that precedes $rd$ in $\sigma_r$. By the code of the *decide* handler, $sm.value$ equals $v'$ at the time when $sm.lastOps[i]$ is set to $\langle n, val\rangle$. Therefore, $val = v'$, $rd$ returns the value that is written by the last *write* operation that precedes it in $\sigma_r$.
- There is no *write* operation in $Req_1$, but there is a request that contains a *write* operation and is chosen before $Req_1$ in $r$. Consider the last such request $Req_2$, and consider the *write* operation $w$ invoked by the process with the lowest id in $Req_2$. Assume that $w$'s argument is $v'$, and $Req_2$ was chosen in a consensus associated with timestamp $TS'$ (notice that $TS' < TS$). By the code of the *decide* handler and Lemma 4, in all the $sm$'s s.t. $sm.ts = TS' + 1$, the value of $sm.value$ is $v'$. Now, since there is no *write* operation in any chosen request between $Req_2$ and $Req_1$ in $r$, no process writes to $sm.value$ when $TS' < sm.ts < TS$. Hence, when $sm.lastOps[i]$ is set to $\langle n, val\rangle$, $sm.value$ equals $v'$, and thus $val = v'$. Therefore, $rd$ returns the value that is written by the last *write* operation that precedes $rd$ in $\sigma_r$.

**Corollary 1.** *Algorithms 1–3 implement an atomic storage service.*

### A.2 Liveness

Consider operation $op_i$ invoked at time $t$ by a correct process $p_i$ in run $r$. Notice that $r$ is a run with either infinitely or finitely many invocations. We show that, in both cases, if $p_i$ is active in $r$, then $op_i$ returns in $r$.

We associate the addition or removal of process $p_j$ by a process $p_i$ with the timestamp that equals $sm_i.ts$ at the time when the operation returns. The addition of all processes in $P_0$ is associated with timestamp 0.

First, we consider runs with infinitely many invocations. In Lemma 8, we show that for every process $p$, every $sm$ associated with a larger timestamp than $p$'s addition contains $p$ in $sm.cng.mem$. In Observation 2, we show that in a run with infinitely many invocations, for every timestamp $ts$, there is a completed operation that has a bigger timestamp than $ts$ at the time of the invocation. Moreover, after the stabilization time of the FD, operations must help all the slow active processes in order to complete. In Lemma 9, we use the observation to show that any operation invoked in a run with infinitely many invocations returns.

Next, we consider runs with finitely many invocations. We show Lemma 10 that eventually all the active members of the last $sm$ adopt it. Then, in Lemma 11, we show that every operation invoked by an active process completes. Finally, Theorem 2, stipulates that the algorithm satisfies wait-freedom.

**Lemma 8.** *Assume the addition of $p_i$ is associated with timestamp $TS$ in run $r$. If $p_i$ is active, then $p_i \in sm.cng.mem$ for every $sm$ s.t. $sm.ts \geq TS$.*

*Proof.* The proof is by induction on $sm.ts$. **Base:** If $p_i \in P_0$, then $p_i \in sm.cng.mem$ for all $sm$ s.t. $sm.ts = 0$. Otherwise, $\langle add, p_i \rangle$ is part of a request that is chosen in a consensus associated with timestamp $TS' = TS - 1$, and thus, by Lemma 4, $p_i \in sm.cng.mem$ for all $sm$ s.t. $sm.ts = TS' + 1 = TS$. **Induction:** Process $p_i$ is active, so no process invokes $\langle remove, p_i \rangle$, and therefore, together with the validity of consensus, no chosen request contains $\langle remove, p_i \rangle$. Hence, if $p_i \in sm.cng.mem$ for $sm$ with $sm.ts = k$, then $p_i \in sm.cng.mem$ for every $sm$ s.t. $sm.ts = k + 1$.

*Claim.* Consider a run $r$ of the algorithm with infinitely many invocations. Then for every time $t$ and timestamp $TS$, there is a completed operation that is invoked after time $t$ by a process with $sm.ts > TS$ at the time of the invocation.

*Proof.* Recall that $r$ is well-formed and only processes in $V(t).join$ can invoke operations at time $t$. Therefore, there are infinitely many completed operations in $r$. Since a finite number of operations are completed with each timestamp, the claim follows.

**Lemma 9.** *Consider an operation $op_i$ invoked at time $t$ by an active process $p_i$ in a run $r$ with infinitely many invocations. Then $op_i$ completes in $r$.*

*Proof.* Assume by way of contradiction that $p_i$ is active and $op_i$ does not complete in $r$. Assume w.l.o.g. that $p_i$'s addition is associated with timestamps $TS$ and $op_i$ is invoked with $opNum_i = n$. Consider a time $t' > t$ after $p_i$ invokes $op_i$ and the FD has stabilized. By Claim A.2, there is a completed operation $op_j$ in $r$, invoked by some process $p_j$ at a time $t'' > t'$ when $sm_j.ts > TS$, whose completion is associated with timestamp $TS'$. By Lemma 8, $p_i \in sm_j.cng.mem$, at time $t''$. Now by the algorithm and by the eventual strong accuracy property of the FD, $p_j$ proposes $op_j$ and $op_i$ in the same request, and continues to propose both of them until one is selected. Note that it is impossible for $op_j$ to be selected without $op_i$ since any process that helps $p_j$ after stabilization

also helps $p_i$. Hence, since $op_j$ completes, they are both performed in the same *decide* handler. The run is well-formed, so $p_i$ does not invoke operations that are associated with $opNum_i > n$. Hence, following the time when $op_i$ is selected, for all $sm$ s.t. $sm.ts > TS'$, $sm.lastOps[i].num = n$. Now, again by Claim A.2, consider a completed operation $op_k$ in $r$, that is invoked by some process $p_k$ at time $t'''$ after the stabilization time of the FD s.t. $sm_k.ts > TS'$ at time $t'''$. Operation $op_k$ cannot complete until $p_i$ receives $p_k$'s $sm$. Therefore, $p_i$ receives $sm$ s.t. $sm.ts \geq TS'$, and thus $sm.lastOps[i].num = n$. Therefore, $p_i$ learns that $op_i$ was performed, and $op_i$ completes. A contradiction.

We now proceed to prove liveness in runs with finitely many invocations.

**Definition 4.** *For every run $r$ of the algorithm, and for any point $t$ in $r$, let $TS_t$ be the timestamp associated with the last consensus that made a decision in $r$ before time $t$. Define $sm^t$, at any point $t$ in $r$, to be the sm's state after the completion of the decide handler associated with timestamp $TS_t$ at any process. By Lemma 4, $sm^t$ is unique. Recall that $sm^0$ is the initial state.*

*Claim.* For every run $r$ of the algorithm, and for any point $t$ in $r$, there is a majority of $sm^t.cng.mem$ $M$ s.t. $M \subseteq (V(t).membership \cup P(t).join) \setminus F(t)$.

*Proof.* By the code of the algorithm, for every run $r$ and for any point $t$ in $r$, $V(t).membership \subseteq sm^t.cng.mem$ and $sm^t.cng.mem \cap V(t).remove = \{\}$. The claim follows from failure condition.

**Observation 2.** *Consider a run $r$ of the algorithm with finitely many invocations. Then there is a point $t$ in $r$ s.t. for every $t' > t$, $sm^t = sm^{t'}$. Denote this $sm$ to be $\hat{sm}$.*

The following lemma follows from Lemma 4, Claim A.2, and the periodic update messages; for space limitations, we omit its proof.

**Lemma 10.** *Consider a run $r$ of the algorithm with finitely many invocations. Then eventually for every active process $p_i \in \hat{sm}.cng.mem$, $sm_i = \hat{sm}$.*

**Lemma 11.** *Consider an operation $op_i$ invoked at time $t$ by an active process $p_i$ in a run $r$ with finitely many invocations. Then $op_i$ completes in $r$.*

*Proof.* By Lemma 8, $p_i \in \hat{sm}.cng.mem$, and by Lemma 10, there is a point $t'$ in $r$ s.t. $sm_i = \hat{sm}$ for all $t \geq t'$. Assume by way of contradiction that $op_i$ does not complete in $r$. Therefore, $op_i$ is either stuck in one of its waits or continuously iterates in a while loop. In each case, we show a contradiction. Denote by *con* the consensus associated with timestamp $\hat{sm}.ts$. By definition of $\hat{sm}$, no decision is made in *con* in $r$.

– Operation $op_i$ waits in line 16 (Algorithm 2) forever. Notice that $\hat{sm}.cng.rem$ contains all the process that were removed in $r$, so, after time $t'$, $p_i$ does not wait for a reply from a removed process. By the strong completeness property of FD, $p_i$ does not wait for faulty processes forever. A contradiction.

- Operation $op_i$ waits in line 18 (Algorithm 2) forever. Notice that from time $t'$ till $p_i$ proposes in $con$, $pend_i = false$. Therefore, $p_i$ proposes in $con$ in line 22 (Algorithm 2), and waits in line 18 after the propose. By Observation 2, there is a majority $M$ of $\hat{sm}.cng.em$ s.t. $M \subseteq V(t).membership \cup P(t).join \setminus F(t)$. Therefore, by the termination of consensus, eventually a decision is made in $con$. A contradiction to the definition of $\hat{sm}$.
- Operation $op_i$ remains in the while loop in line 17 (Algorithm 2) forever. Since it does not waits in line 18 (Algorithm 2) forever, $op_i$ proposes infinitely many times, and since each propose is made in a different consensus and $p_i$ can propose in a consensus beyond the first one only once a decision is made in the previous one, infinitely many decisions are made in $r$. A contradiction to the definition of $\hat{sm}$.
- Operation $op_i$ waits in line 27 (Algorithm 2) forever. Consider two cases. First, $sm_i \neq \hat{sm}$ when $p_i$ performs line 26 (Algorithm 2). In this case, $p_i$ continues at time $t'$, when it adopts $\hat{sm}$, because $sm_i.ts > ts$ hold at time $t'$. In the second case ($sm_i = \hat{sm}$ when $p_i$ performs line 26), $p_i$ sends *update* message to all processes in $\hat{sm}.cng.mem$, and waits for a majority to reply. By Observation 2, there is a correct majority in $\hat{sm}.cng.mem$, and thus $p_i$ eventually receives the replies and continues. In both cases we have contradiction.

Therefore, $p_i$ completes in $r$.

We conclude with the following theorem:

**Theorem 2.** *Algorithms 1–3 implement wait-free atomic dynamic storage.*

## References


1. Aguilera, M.K., Keidar, I., Malkhi, D., Martin, J.P., Shraer, A., et al.: Reconfiguring replicated atomic storage: a tutorial. Bull. EATCS **102**, 84–108 (2010)
2. Aguilera, M.K., Keidar, I., Malkhi, D., Shraer, A.: Dynamic atomic storage without consensus. J. ACM **58**(2), 7 (2011)
3. Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message-passing systems. J. ACM (JACM) **42**(1), 124–142 (1995)
4. Attiya, H., Chung, H.C., Ellen, F., Kumar, S., Welch, J.L.: Simulating a shared register in an asynchronous system that never stops changing. In: Moses, Y. (ed.) DISC 2015. LNCS, vol. 9363, pp. 75–91. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48653-5_6
5. Baldoni, R., Bonomi, S., Kermarrec, A.M., Raynal, M.: Implementing a register in a dynamic distributed system. In: 29th IEEE International Conference on Distributed Computing Systems, ICDCS 2009, pp. 639–647. IEEE (2009)
6. Baldoni, R., Bonomi, S., Raynal, M.: Regular register: an implementation in a churn prone environment. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 15–29. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11476-2_3
7. Baldoni, R., Bonomi, S., Raynal, M.: Implementing a regular register in an eventually synchronous distributed system prone to continuous churn. IEEE Trans. Parallel Distrib. Syst. **23**(1), 102–109 (2012)

8. Birman, K., Malkhi, D., Van Renesse, R.: Virtually synchronous methodology for dynamic service replication (2010)
9. Chockler, G., Gilbert, S., Gramoli, V., Musial, P.M., Shvartsman, A.A.: Reconfigurable distributed storage for dynamic networks. J. Parallel Distrib. Comput. **69**(1), 100–116 (2009)
10. Chockler, G.V., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Comput. Surv. (CSUR) **33**(4), 427–469 (2001)
11. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
12. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
13. Gafni, E., Malkhi, D.: Elastic configuration maintenance via a parsimonious speculating snapshot solution. In: Moses, Y. (ed.) DISC 2015. LNCS, vol. 9363, pp. 140–153. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48653-5_10
14. Gilbert, S., Lynch, N., Shvartsman, A.: RAMBO II: rapidly reconfigurable atomic memory for dynamic networks. In: DSN. IEEE Computer Society (2003)
15. Gilbert, S., Lynch, N.A., Shvartsman, A.A.: RAMBO: a robust, reconfigurable atomic memory service for dynamic networks. Distrib. Comput. **23**(4), 225–272 (2010)
16. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. **12**(3), 463–492 (1990)
17. Jehl, L., Meling, H.: The case for reconfiguration without consensus. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing. ACM (2016)
18. Jehl, L., Vitenberg, R., Meling, H.: SmartMerge: a new approach to reconfiguration for atomic storage. In: Moses, Y. (ed.) DISC 2015. LNCS, vol. 9363, pp. 154–169. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48653-5_11
19. Ko, S.Y., Hoque, I., Gupta, I.: Using tractable and realistic churn models to analyze quiescence behavior of distributed protocols. In: IEEE Symposium on Reliable Distributed Systems, SRDS 2008, pp. 259–268. IEEE (2008)
20. Lamport, L.: On interprocess communication. Distrib. Comput. **1**(2), 86–101 (1986)
21. Lamport, L., Malkhi, D., Zhou, L.: Reconfiguring a state machine. ACM SIGACT News **41**(1), 63–73 (2010)
22. Lin, K., Hadzilacos, V.: Asynchronous group membership with oracles. In: Jayanti, P. (ed.) DISC 1999. LNCS, vol. 1693, pp. 79–94. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48169-9_6
23. Lynch, N., Shvartsman, A.A.: RAMBO: a reconfigurable atomic memory service for dynamic networks. In: Malkhi, D. (ed.) DISC 2002. LNCS, vol. 2508, pp. 173–190. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36108-1_12
24. Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., Abbadi, A.E.: From static distributed systems to dynamic systems. In: 24th IEEE Symposium on Reliable Distributed Systems, SRDS 2005, pp. 109–118. IEEE (2005)
25. Shraer, A., Martin, J.P., Malkhi, D., Keidar, I.: Data-centric reconfiguration with network-attached disks. In: LADIS 2010 (2010)
26. Spiegelman, A., Keidar, I., Malkhi, D.: Dynamic reconfiguration: a tutorial. In: OPODIS (2015)
27. Spiegelman, A., Keidar, I., Malkhi, D.: Dynamic reconfiguration: abstraction and optimal asynchronous solution. In: DISC (2017)

# Author Index