Guy Even
Magnús M. Halldórsson (Eds.)

Structural Information and Communication Complexity

LNCS 7355

19th International Colloquium, SIROCCO 2012
Reykjavik, Iceland, June/July 2012
Proceedings

Springer

Lecture Notes in Computer Science 7355

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

Guy Even Magnús M. Halldórsson (Eds.)

Structural Information and Communication Complexity

19th International Colloquium, SIROCCO 2012
Reykjavik, Iceland, June 30 – July 2, 2012
Proceedings

Springer

Volume Editors

Guy Even
Tel-Aviv University
School of Electrical Engineering
Ramat-Aviv, Tel Aviv 69978, Israel
E-mail: guy@eng.tau.ac.il

Magnús M. Halldórsson
Reykjavík University
ICE-TCS, School of Computer Science
101 Reykjavík, Iceland
E-mail: mmh@ru.is

ISSN 0302-9743
ISBN 978-3-642-31103-1
DOI 10.1007/978-3-642-31104-8

e-ISSN 1611-3349
e-ISBN 978-3-642-31104-8

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012939376

CR Subject Classification (1998): F.2, C.2, G.2, E.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2012) took place in Reykjavik, Iceland, for three days starting June 30, 2012.

SIROCCO is devoted to the study of communication and knowledge in distributed systems from both qualitative and quantitative viewpoints. Special emphasis is given to innovative approaches and fundamental understanding in addition to efforts to optimize current designs. The typical areas include distributed computing, communication networks, game theory, parallel computing, social networks, mobile computing (including autonomous robots), peer-to-peer systems, communication complexity, fault-tolerant graph theories, and randomized/probabilistic issues in networks.

This year, 54 papers were submitted in response to the call for papers, and each paper was evaluated by three to five reviewers. In total, 164 reviews were received. The Program Committee selected 28 papers for presentation at the colloquium and publication in this volume after in-depth discussions.

The SIROCCO Prize for Innovation in Distributed Computing was awarded this year to Roger Wattenhofer from ETH Zurich. A commendation summarizing his many and important innovative contributions to distributed computing appears in these proceedings.

The collaboration of the Program Committee members and the external reviewers enabled completing the process of reviewing the papers and discussing them in less than four weeks. We thank them all for their devoted service to the SIROCCO community. We thank the authors of all the submitted papers; without them we could not have prepared a program of such quality. We thank Yvonne Anne Pignolet for her assistance as Publicity Chair.

We thank the invited speakers Roger Wattenhofer and Boaz Patt-Shamir.

The preparation of this event was guided by the SIROCCO Steering Committee, headed by Shay Kutten.

We are grateful to Reykjavik University for hosting the meeting. In particular, we thank the Events and Facilities Departments of the university for their support.

The EasyChair system was used to handle the submission of papers, manage the review process, and generate these proceedings.

April 2012

Guy Even
Magnús M. Halldórsson

Prize for Innovation in Distributed Computing Awarded to Roger Wattenhofer

The Prize for Innovation in Distributed Computing for 2012 is awarded to Roger Wattenhofer (ETH Zurich), in recognition of his extensive contributions to the study of distributed approximation, as illustrated by papers that have appeared in the proceedings of past SIROCCO meetings.

We requested David Peleg to write the following laudatio for the proceedings, describing Wattenhofer's contributions.

Wattenhofer explored aspects of distributed approximation in a variety of distributed communication models and classes of underlying networks, seeking to form a detailed picture of the approximable and the inapproximable in ordinary (wired) as well as wireless networks. He also developed the strongest known distributed approximation algorithms and inapproximability results for some of the central problems in the area.

Broadly speaking, one can classify Wattenhofer's achievements in the area of distributed approximation into several main contributions, described next.

Initiating the study of distributed approximation for key problems. Wattenhofer *initiated* the study of distributed approximation (as well as *distributed inapproximability*) for a number of central problems. A notable example is the problem of *facility location*, which he studied in his PODC'05 paper with Moscibroda [7], exploring a trade-off between the amount of communication and the resulting approximation ratio.

LP-based distributed approximation. Wattenhofer significantly advanced and popularized the use of *LP-based* approximation techniques in the distributed domain, including techniques based on *randomized rounding of fractional relaxations* of given integer linear programs, *LP duality*, and more. For instance, his paper with Fabian Kuhn in PODC'03 [1] used LP relaxation techniques to yield a constant-time distributed algorithm for approximating the *minimum dominating set* (MDS) problem. It was the first distributed MDS approximation algorithm that achieves a nontrivial approximation ratio in a constant number of rounds. Another example for a result derived via a technique based on a distributed primal-dual approach for approximating a linear program is his paper [7] on facility location, discussed above.

Local distributed approximation. Wattenhofer focused attention on *local* distributed approximation, namely, distributed approximation algorithms that operate in *constant time*, and established a number of strong lower bounds on the approximation ratio achievable by such algorithms, thus contributing significantly to our understanding of locality. Two of his papers neatly illustrate this type of work. One is his PODC'06 paper with Kuhn on the complexity of distributed graph coloring [2], which considers coloring algorithms that run for a

single communication round. The second is his PODC'04 paper with Kuhn and Moscibroda [3], which gives time lower bounds for the distributed approximation of *minimum vertex cover* (MVC) and minimum dominating set (MDS). This paper shows that an analog to the well-known greedy algorithm for approximating MVC within a factor 2 does not exist in the distributed local setting.

Relationships with relevant graph parameters. Wattenhofer explored the dependencies of approximability and inapproximability of central problems, such as coloring, MIS, and minimum dominating set (MDS), on relevant graph parameters. This contribution is illustrated by his most recent SIROCCO paper on distributed approximation, published in SIROCCO'11, together with Schneider [10]. This paper explores the dependencies of the complexity of distributed approximate coloring on the spectrum of maximum neighborhood sizes and the chromatic number of the graph at hand.

The above description covers just a sample from Wattenhofer's impressive list of results on distributed approximation. Some of his other notable contributions in this area are [4–6, 8, 9, 11].

Wattenhofer has published extensively in SIROCCO. In addition to [10], discussed above, he also published five strong papers in other areas of distributed computing and networking, unrelated to distributed approximation, including in particular rumor dissemination [13], sensor networks [14], routing [15], Peer to Peer networks [16], and distributed counting [17].

In summary, Wattenhofer's substantial and extensive work in the area of distributed approximation, along with his many other contributions to the field of distributed network algorithms at large, helped galvanizing the field and reviving it with new problems, refreshing viewpoints and novel powerful techniques.

2012 Prize Committee

Evangelos Kranakis	Carleton University, Canada
Shay Kutten	Technion, Israel (Chair)
Alexander A. Shvartsman	University of Connecticut, USA
Boaz Patt-Shamir	Tel Aviv University, Israel
Masafumi Yamashita	Kyushu University, Japan

References

1. Kuhn, F., Wattenhofer, R.: Constant-time distributed dominating set approximation. In: Proc. PODC, pp. 25–32 (2003)
2. Kuhn, F., Wattenhofer, R.: On the complexity of distributed graph coloring. In: Proc. PODC, pp. 7–15 (2006)
3. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proc. PODC, pp. 300–309 (2004)
4. Lenzen, C., Oswald, Y.-A., Wattenhofer, R.: What can be approximated locally? Case study: Dominating sets in planar graphs. In: Proc. SPAA, pp. 46–54 (2008)
5. Lenzen, C., Wattenhofer, R.: Leveraging Linial's Locality Limit. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 394–407. Springer, Heidelberg (2008)

6. Lenzen, C., Wattenhofer, R.: Minimum Dominating Set Approximation in Graphs of Bounded Arboricity. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 510–524. Springer, Heidelberg (2010)
7. Moscibroda, T., Wattenhofer, R.: Facility location: Distributed approximation. In: Proc. PODC, pp. 108–117 (2005)
8. Schneider, J., Wattenhofer, R.: Coloring unstructured wireless multi-hop networks. In: Proc. PODC, pp. 210–219 (2009)
9. Schneider, J., Wattenhofer, R.: Trading Bit, Message, and Time Complexity of Distributed Algorithms. In: Peleg, D. (ed.) Distributed Computing. LNCS, vol. 6950, pp. 51–65. Springer, Heidelberg (2011)
10. Schneider, J., Wattenhofer, R.: Distributed Coloring Depending on the Chromatic Number or the Neighborhood Growth. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 246–257. Springer, Heidelberg (2011)
11. Wattenhofer, M., Wattenhofer, R.: Distributed Weighted Matching. In: Guerraoui, R. (ed.) DISC 2004. LNCS, vol. 3274, pp. 335–348. Springer, Heidelberg (2004)
12. van Wattenhofer, M., Wattenhofer, R.: Fast and Simple Algorithms for Weighted Perfect Matching. In: Proc. CTW, pp. 246–252 (2004)
13. Kostka, J., Oswald, Y.A., Wattenhofer, R.: Word of Mouth: Rumor Dissemination in Social Networks. In: Shvartsman, A.A., Felber, P. (eds.) SIROCCO 2008. LNCS, vol. 5058, pp. 185–196. Springer, Heidelberg (2008)
14. Wattenhofer, R.: Sensor Networks: Distributed Algorithms Reloaded – or Revolutions? In: Flocchini, P., Gąsieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 24–28. Springer, Heidelberg (2006)
15. Wattenhofer, M., Wattenhofer, R., Widmayer, P.: Geometric Routing Without Geometry. In: Pelc, A., Raynal, M. (eds.) SIROCCO 2005. LNCS, vol. 3499, pp. 307–322. Springer, Heidelberg (2005)
16. Giesen, J., Wattenhofer, R., Zollinger, A.: Towards a Theory of Peer-to-Peer Computability. In: Proc. SIROCCO, pp. 115–132 (2002)
17. Wattenhofer, R., Widmayer, P.: The counting pyramid: an adaptive distributed counting scheme. In: Proc. SIROCCO, pp. 145–157 (1998)

Organization

Program Committee

Dimitris Achlioptas	University of California, Santa Cruz, USA
James Aspnes	Yale, USA
Nikhil Bansal	Eindhoven University of Technology, The Netherlands
Thomas Erlebach	University of Leicester, UK
Guy Even	Tel Aviv University, Israel
Magnús M. Halldórsson	Reykjavik University, Iceland
Fabian Kuhn	University of Lugano, Switzerland
Stefano Leonardi	University of Rome "La Sapienza", Italy
Toshimitsu Masuzawa	Osaka University, Japan
Friedhelm Meyer auf der Heide	Heinz Nixdorf Institute, Germany
Sriram Pemmaraju	The University of Iowa, USA
Harald Räcke	Technical University of Munich, Germany
Ignasi Sau	CNRS, LIRMM, France
Thomas Sauerwald	Simon Fraser University, Canada
Christian Schindelhauer	University of Freiburg, Germany
Hadas Shachnai	Technion, Israel
Christian Sohler	Technical University of Dortmund, Germany
Subhash Suri	University of California, Santa Barbara, USA
Philippas Tsigas	Chalmers University, Sweden
Shmuel Zaks	Technion, Israel

Steering Committee

Ralf Klasing	CNRS and University of Bordeaux, France
Shay Kutten	Technion, Israel (Chair)
Boaz Patt-Shamir	Tel Aviv University, Israel
Alex Shvartsman	MIT and University of Connecticut, USA
Masafumi Yamashita	Kyushu University, Japan

Honorary Board

Paola Flocchini	University of Ottawa, Canada
Pierre Fraigniaud	CNRS and University Paris Diderot, France
Lefteris Kirousis	University of Patras, Greece
Rastislav Kralovic	Comenius University, Slovakia
Evangelos Kranakis	Carleton University, Canada
Danny Krizanc	Wesleyan University, USA

Bernard Mans	Macquarie University, Australia
David Peleg	Weizmann Institute, Israel
Nicola Santoro	Carleton University, Canada
Pavlos Spirakis	CTI, Greece

Local Arrangements Chair

Magnús M. Halldórsson	Reykjavik University, Iceland

Publicity Chair

Yvonne Anne Pignolet	ABB Corporate Research

Additional Reviewers

Akbari, Hoda
Anagnostopoulos, Aris
Applebaum, Benny
Bampas, Evangelos
Becchetti, Luca
Berns, Andrew
Bollig, Beate
Bonifaci, Vincenzo
Chatterjee, Bapi
Cheilaris, Panagiotis
Cord-Landwehr, Andreas
Crowston, Robert
Flammini, Michele
Gamzu, Iftah
Gasieniec, Leszek
Giroire, Frederic
Hay, David
Hegeman, James
Hoefer, Martin
Kakugawa, Hirotsugu
Kempkes, Barbara
Klasing, Ralf
Kling, Peter
Kniesburges, Sebastian
Kowalik, Lukasz
Larsson, Andreas
Lee, Kayi
van Leeuwen, Erik Jan
Margaliot, Michael
Matsri, Yakov
McGregor, Andrew
Medina, Moti
Megow, Nicole
Mertzios, George
Mitra, Pradipta
Mnich, Matthias
Nagarajan, Viswanath
Ooshita, Fukuhito
Pelc, Andrzej
Pourmiri, Ali
Prencipe, Giuseppe
Pruhs, Kirk
Rawitz, Dror
Sampaio, Leonardo
Sardeshmukh, Vivek
Shahar, Moni
Shalom, Mordechai
Skopalik, Alexander
Smorodinsky, Shakhar
Souza, Alexander
Tamir, Tami
Woelfel, Philipp
Wong, Prudence W.H.

Distributed Complexity Theory

Roger Wattenhofer
Distributed Computing (DISCO)
ETH Zurich, 8092 Zurich, Switzerland
wattenhofer@ethz.ch

In the last decades our community has made tremendous progress towards understanding the complexity of distributed message passing algorithms. Given a network with n nodes and diameter D, we managed to established a rich selection of upper and lower bounds regarding how much time it takes to solve or approximate a problem. Currently we know five main distributed complexity classes:

- Strictly *local* problems can be solved in constant $\Theta(1)$ time, e.g. a constant approximation of a dominating set in a planar graph.
- Just a little bit slower are problems that can be solved in *log-star* $\Theta(\log^* n)$ time, e.g. many combinatorial optimization problems in special graph classes such as growth bounded graphs.
- A large body of problems is *polylogarithmic* (or *pseudo-local*), in the sense that they seem strictly local but are not, as they need $\Theta(\text{polylog } n)$ time, e.g. the maximal independent set problem.
- There are problems which are *global* and need $\Theta(D)$ time, e.g. to count the number of nodes in the network.
- Finally there are problems which need *polynomial* $\Theta(\text{poly } n)$ time, even if the diameter D is a constant, e.g. computing the diameter of the network.

In my talk I will introduce the message passing model, present a few selected results, mention prominent open problems, and discuss some of the most exciting future research directions.

Acknowledgments: I would like thank my former or current students for contributing significantly to the core of this theory, in chronological order: Fabian Kuhn, Thomas Moscibroda, Yvonne Anne Pignolet, Christoph Lenzen, Johannes Schneider, and Stephan Holzer.

No Piece Missing: Online Set Packing

Boaz Patt-Shamir
School of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel

Abstract. We consider a scenario where large data frames are broken into a few packets and transmitted over the network. Our focus is on a bottleneck router: the model assumes that in each time step, a set of packets (a burst) arrives, from which only one packet can be served, and all other packets are lost. A data frame is considered useful only if none of its constituent packets is lost, and otherwise it is worthless. We abstract the problem as a new type of *online set packing*, present a randomized distributed algorithm and a nearly-matching lower bound on the competitive ratio for any randomized online algorithm. We show how the basic approach extends to various models, including cases with redundancy, buffering and others.
This talk is based on various papers co-authored with Yuval Emek, Magnus Halldòrsson, Yishay Mansour, Jaikumar Radhakrishnan and Dror Rawitz.

Table of Contents

Space Lower Bounds for Low-Stretch Greedy Embeddings*

Ioannis Caragiannis and Christos Kalaitzis

Computer Technology Institute and Press "Diophantus" &
Department of Computer Engineering and Informatics,
University of Patras, 26504 Rio, Greece

Abstract. Greedy (geometric) routing is an important paradigm for routing in communication networks. It uses an embedding of the nodes of the network into points of a metric space (e.g., $\mathbb{R}^d$) equipped with a distance function (e.g., the Euclidean distance ℓ_2) and uses as address of each node the coordinates of the corresponding point. The embedding has particular properties so that the routing decision for a packet is taken greedily based only on the addresses of the current node, its neighbors, and the destination node and the distances of the corresponding points. In this way, there is no need to keep routing tables at the nodes. Embeddings that allow for this functionality are called greedy embeddings. Even though greedy embeddings do exist for several metric spaces and distance functions, they usually result in paths of high stretch, i.e., significantly longer than the corresponding shortest paths.

In this paper, we show that greedy embeddings in low-dimensional Euclidean spaces necessarily have high stretch. In particular, greedy embeddings of n-node graphs with optimal stretch requires at least $\Omega(n)$ dimensions for distance ℓ_2. This result disproves a conjecture by Maymounkov (2006) stating that greedy embeddings of optimal stretch are possible in Euclidean spaces with polylogarithmic number of dimensions. Furthermore, we present trade-offs between the stretch and the number of dimensions of the host Euclidean space. Our results imply that every greedy embedding into a Euclidean space with polylogarithmic number of dimensions (and Euclidean distance) has stretch $\Omega\left(\frac{\log n}{\log\log n}\right)$. We extend this result for a distance function used by an $O(\log n)$-stretch greedy embedding presented by Flury, Pemmaraju, and Wattenhofer (2009). Our lower bound implies that their embedding has almost best possible stretch.

1 Introduction

Greedy routing utilizes a particular assignment of node addresses so that routing of packets can be performed using only the address of the current node of a traveling packet, the addresses of its neighbors, and the address of the destination node. The node addresses are usually defined using a *greedy embedding*. Formally,

* This work was partially supported by the EC-funded STREP project EULER.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 1–12, 2012.

a greedy embedding of a graph $G = (V, E)$ into a host metric space (X, dist) is a function $f : V \to X$ so that the following property holds: for any two nodes u, t of G, there exists a node v in the neighborhood $\Gamma(u)$ of u in G so that $\text{dist}(f(v), f(t)) < \text{dist}(f(u), f(t))$. A typical example of a host metric space is the d-dimensional Euclidean space $\mathbb{R}^d$ equipped with the Euclidean distance ℓ_2. Given a greedy embedding f, the coordinates of point $f(u)$ can be used as the address of node u. Then, when node u has to take a decision about the next hop for a packet with destination address $f(t)$, it has to select among its neighbors a node with address $f(v)$ such that $\text{dist}(f(v), f(t)) < \text{dist}(f(u), f(t))$. It is clear that, in this way, the packet is guaranteed to reach its destination within a finite number of steps.

Greedy embeddings were first defined by Papadimitriou and Ratajczak [15]. They proved that any 3-connected planar graph can be greedily embedded into the 3-dimensional Euclidean space using a non-Euclidean distance function. They also conjectured that every such graph can be greedily embedded in the Euclidean space with Euclidean distance. The conjecture was proved by Moitra and Leighton [14]. Kleinberg [10] showed that any tree can be greedily embedded in the 2-dimensional hyperbolic space. This immediately yields a greedy embedding for any graph (by just embedding a spanning tree). Epstein and Woodrich [6] observed that the coordinates of the nodes in Kleinberg's embedding require too much space and modified it so that each coordinate is represented with $O(\log n)$ bits (where n is the size of the graph). Greedy embeddings into $O(\log n)$-dimensional Euclidean spaces (with ℓ_∞ distance) are also known [10] and exploit an isometric embedding of trees into Euclidean spaces due to Linial et al. [11].

Note that the approach of computing the greedy embedding of a spanning tree ignores several links of the network. Hence, it may be the case that even though a packet could potentially reach its destination with a few hops using a shortest path, it is greedily routed through a path that has to travel across a constant fraction of the nodes of the whole network. The measure that can quantify this inefficiency is the stretch of a greedy routing algorithm, i.e., the ratio between the length of the path used by the algorithm over the length of the corresponding shortest path.

Let us proceed with a formal definition of the stretch of a greedy embedding.

Definition 1. *Let f be a greedy embedding of a graph G into a metric space $(X, dist)$. A path $\langle u_0, u_1, ..., u_t \rangle$ is called a greedy path if $u_{i+1} \in argmin_{v \in \Gamma(u_i)}\{dist(f(v), f(u_t))\}$, where $\Gamma(u_i)$ denotes the neighborhood of u_i in G. We say that f has stretch ρ for graph G if, for every pair of nodes u, v of G, the length of every greedy path from u to v is at most ρ times the length of the shortest path from u to v in G. The stretch of f is simply the maximum stretch over all graphs.*

We use the terms no-stretch and optimal stretch to refer to embeddings with stretch equal to 1.

Maymounkov [13] considers the question of whether no-stretch greedy embeddings into low-dimensional spaces exist. Among other results, he presents a lower

bound of $\Omega(\log n)$ on the dimension of the host hyperbolic space for greedy embeddings with optimal stretch. Furthermore, he conjectures that any graph can be embedded into Euclidean or hyperbolic spaces with a polylogarithmic number of dimensions with no stretch. We remark that a proof of this conjecture would probably justify greedy routing as a compelling alternative to compact routing [16].

Flury et al. [8] present a greedy embedding of any n-node graph into an $O(\log^2 n)$-dimensional Euclidean space that has stretch $O(\log n)$. Each coordinate in their embedding uses $O(\log n)$ bits. They used the min-max$_c$ distance function which views the d-dimensional Euclidean space as composed by d/c c-dimensional spaces and, for a pair of points x, y, takes the ℓ_∞ norm of the projections of x and y into those spaces, and finally takes the minimum of those ℓ_∞ distances as the min-max$_c$ distance between them. Their embedding uses an algorithm of Awerbuch and Peleg [4] to compute a tree cover of the graph and the algorithm of Linial et al. [11] to embed each tree in the cover isometrically in a low-dimensional Euclidean space. In practice, greedy embeddings have been proved useful for sensor [8] and internet-like networks [5].

In this paper, we present lower bounds on the number of dimensions required for low-stretch greedy embeddings into Euclidean spaces. We first disprove Maymounkov's conjecture by showing that greedy embeddings into $(\mathbb{R}^d, \ell_2)$ have optimal stretch only if the number of dimensions d is linear in n. The proof uses an extension of the hard crossroads construction in [13] and exploits properties of random sign pattern matrices. Namely, we make use of a linear lower bound due to Alon et al. [3] on the minimum rank of random $N \times N$ sign pattern matrices. We also obtain an $\Omega(\sqrt{n})$ lower bound through an explicit construction that uses Hadamard matrices. These results are stated in Theorem 2.

Furthermore, we present trade-offs between the stretch of greedy embeddings into $\mathbb{R}^d$ and the number of dimensions d for different distance functions. Namely, for every integer parameter $k \geq 3$, we show that greedy embeddings into $\mathbb{R}^d$ with ℓ_p distance have stretch smaller than $\frac{k+1}{3}$ only if $d \in O\left(\frac{n^{1/k}}{\log p}\right)$ (Theorem 5). This implies that the best stretch we can expect with a polylogarithmic number of dimensions is $\Omega\left(\frac{\log n}{\log \log n}\right)$. Our arguments use a result of Erdös and Sachs [7] on the density of graphs of high girth and a result of Warren [17] that upper-bounds the number of different sign patterns of a set of polynomials. In particular, starting from a dense graph with high girth, we construct a family of graphs and show that, if d is not sufficiently large, any embedding f fails to achieve low stretch for some graph in this family.

We extend our lower bound arguments to greedy embeddings into $\mathbb{R}^d$ that use the min-max$_c$ distance function that has been used in [8]. Note that the lower bound does not depend on any other characteristic of the embedding of [8] and applies to every embedding in $(\mathbb{R}^d, \text{min-max}_c)$. We show that the best stretch we can hope for with $d \in \text{polylog}(n)$ is $\Omega\left(\frac{\log n}{\log \log n}\right)$ (Theorem 7). This lower bound indicates that the embedding of Flury et al. [8] is almost optimal among

all greedy embeddings in $(\mathbb{R}^d, \text{min-max}_c)$. Furthermore, our proof applies to a larger class of distance functions including ℓ_∞.

We remark that our proofs are not information-theoretic in the sense that we do not prove lower bounds on the number of bits required in order to achieve low stretch embeddings. Instead, we prove that the particular characteristics of the host metric space (e.g., small number of dimensions) and the distance function do not allow for low stretch greedy embeddings even if we allow for arbitrarily many bits to store the coordinates.

We also remark that there is an extensive literature on low-distortion embeddings where the objective is to embed a metric space (e.g., a graph with the shortest path distance function) into another metric space so that the distances between pairs of points are distorted as less as possible (see [12] for an introduction to the subject and a coverage of early related results). Greedy embeddings are inherently different than low-distortion ones. Ordinal embeddings (see [2]) where one aims to maintain the relative order of intra-point distances are conceptually closer to our work. Actually, the use of Warren's theorem has been inspired by [2].

The rest of the paper is structured as follows. We present the extension of the hard crossroads construction in Section 2. In Section 3, we present the trade-off for greedy embeddings into Euclidean spaces using the distance function ℓ_p. The optimality of the embedding of Flury et al. [8] is proved in Section 4. We conclude in Section 5.

2 Hard Crossroads

In this section we extend the hard crossroads construction of Maymounkov [13] by exploiting sign pattern matrices. We begin with some necessary definitions. Throughout the text, we use $[N]$ to denote the set $\{1, 2, ..., N\}$. We also make use of the signum function $\text{sgn} : \mathbb{R} \setminus \{0\} \to \{-1, 1\}$.

Definition 2. *A square $N \times N$ matrix S is called a sign-pattern matrix if $S_{i,j} \in \{-1, 1\}$ for every $i, j \in [N]$. The minimum rank of a sign-pattern matrix S, denoted by $mr(S)$, is the minimum rank among all $N \times N$ matrices M with non-zero entries for which it holds that $sgn(M_{i,j}) = S_{i,j}$ for every $i, j \in [N]$.*

In our construction, we use sign-pattern matrices of high minimum rank. Such matrices do exist as the following result of Alon et al. [3] indicates.

Theorem 1 (Alon, Frankl, and Rödl [3]). *For every integer $N \geq 1$, there exist $N \times N$ sign-pattern matrices of minimum rank at least $N/32$.*

The proof of this statement uses the probabilistic method. As such, the result is existential and does not provide an explicit construction (for many different values of N). A slightly weaker lower bound on the minimum rank is obtained by the so-called Hadamard matrices.

Definition 3. *A Hadamard matrix H_N with N nodes (where N is a power of 2) is defined as $H_1 = [1]$ and*

$$H_N = \begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix}.$$

It is known (see [9]) that the sign-pattern matrix H_N has $\mathrm{mr}(H_N) \geq \sqrt{N}$.

We are ready to present our hard crossroads construction. We will use an $N \times N$ sign pattern matrix S. Given S, we construct the hard crossroads graph $G(S)$ as follows. G has three levels of nodes:

- There are N nodes at level 0: u_1, u_2, ..., u_N.
- There are $2N$ nodes at level 1: nodes u_i^+ and u_i^- for each node u_i of level 0.
- There are also N nodes v_1, ..., v_N at level 2.

The set of edges of $G(S)$ is defined as follows. Each node u_i is connected to both nodes u_i^+ and u_i^- at level 1. For each pair of integers $1 \leq i, j \leq N$, there is an edge between u_i^+ and v_j if $S_{ij} = 1$ and an edge between u_i^- and v_j otherwise. See Figure 1 for an example that uses the sign pattern matrix

$$S = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix}.$$

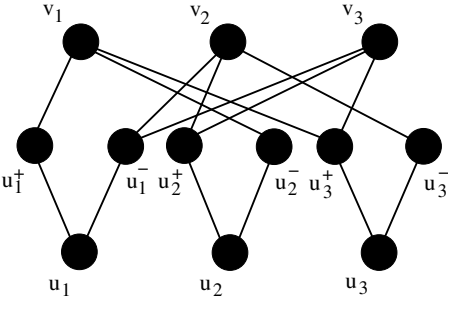


Fig. 1. An example of a hard crossroad

We will show that any greedy embedding f of $G(S)$ into $\mathbb{R}^d$ with distance function ℓ_2 has stretch smaller than 2 only if d is large. Observe that, for every pair of integers i, j, the unique shortest path (of length 2) from node u_i to node v_j goes either through node u_i^+ (if $S_{ij} = 1$) or node u_i^- (if $S_{ij} = -1$); any other path connecting these two nodes has length at least 4. Since f has stretch smaller than 2, this means that the only greedy path from node u_i to node v_j defined by f is their shortest path. Hence, f should satisfy that $\mathrm{dist}(f(u_i^+), f(v_j)) < \mathrm{dist}(f(u_i^-), f(v_j))$ if $S_{i,j} = 1$ and $\mathrm{dist}(f(u_i^+), f(v_j)) > \mathrm{dist}(f(u_i^-), f(v_j))$ otherwise. In other words, depending on whether $S_{i,j} = 1$ or $S_{i,j} = -1$, the embedding

should guarantee that the point $f(v_j)$ lies at the "left" or the "right" side of the hyperplane that bisects points $f(u_i^+)$ and $f(u_i^-)$.

For $i \in [N]$, denote by $U_i^+, U_i^- \in \mathbb{R}^d$ the vectors of coordinates of points $f(u_i^+)$ and $f(u_i^-)$. Also, for $j \in [N]$, denote by $V_j \in \mathbb{R}^d$ the vector of coordinates of point $f(v_j)$. Finally, denote by $(a_i, b_i) = \{x \in \mathbb{R}^d : a_i^T x = b_i\}$ the hyperplane that bisects points $f(u_i^+)$ and $f(u_i^-)$. Without loss of generality, assume that $a_i^T U_i^+ - b_i > 0$ (and $a_i^T U_i^- - b_i < 0$).

By the property of the greedy embedding f, the hyperplane (a_i, b_i) partitions the set of points corresponding to nodes of level 2 into two sets depending on whether node v_j is connected to u_i^+ (i.e., when $S_{i,j} = 1$) or u_i^- (when $S_{i,j} = -1$). We have $a_i^T V_j - b_i > 0$ for each $i, j \in [N]$ such that $S_{i,j} = 1$ and $a_i^T V_j - b_i < 0$, otherwise.

Now, denote by V the $d \times N$ matrix with columns $V_1, V_2, ..., V_N$, by A the $N \times d$ matrix with rows $a_1^T, ..., a_N^T$, and by b the vector with entries $b_1, ..., b_N$. Let $M = A \cdot V - b \cdot 1^T$. Observe that $\mathrm{sgn}(M_{i,j}) = S_{i,j}$, for every $i, j \in [N]$ and, hence, $r(M) \geq \mathrm{mr}(S)$. Using the fact that d is not smaller than the rank of matrix V and well-known facts about the rank of matrices, we obtain

$$d \geq \mathrm{rank}(V) \geq \mathrm{rank}(A \cdot V) = \mathrm{rank}(M + b \cdot 1^T) \geq \mathrm{rank}(M) - 1 \geq \mathrm{mr}(S) - 1.$$

The next statement follows using either a sign pattern matrix with minimum rank at least $N/32$ (from Theorem 1, such graph do exist) or the Hadamard matrix with N nodes (and by observing that the number of nodes in $G(S)$ is $n = 4N$).

Theorem 2. *Let f be a greedy embedding of n-node graphs into the metric space $\mathbb{R}^d$ with distance function ℓ_2. If f has stretch smaller than 2, then $d \geq n/128 - 1$. Furthermore, for every value of n that is a power of 2, there exists an explicitly constructed n-node graph which cannot be embedded into $(\mathbb{R}^d, \ell_2)$ with stretch smaller than 2 if $d < \sqrt{n}/2 - 1$.*

3 A Lower Bound Based on High-Girth Graphs

Our second lower bound argument exploits graphs with high girth. The main observation behind the construction described below is that, in a graph of girth g, if the greedy path between two very close nodes (say, of original distance 2) defined by an embedding is not the corresponding shortest path, then this embedding has stretch at least $\Omega(g)$.

Given a girth-g graph $G = (V, E)$ with N nodes and m edges, we will construct a family $\mathcal{H}$ of 4^m graphs so that every greedy embedding in a low-dimensional Euclidean space with distance ℓ_p has high stretch for some member of this family.

The family $\mathcal{H}$ is constructed as follows. Each graph in this family has a supernode u of three nodes u_0, u_1, and u_2 for each node u of G. Denote by $n = 3N$ the number of nodes of the graphs in $\mathcal{H}$. Node u_0 is connected with nodes u_1 and u_2. For each edge (u, v) of G, there are four different ways to connect the supernodes u and v: one of the nodes u_1 and u_2 is connected to one of the nodes

v_1 and v_2. See Figure 2 for an example. Note that the 4^m different graphs of the family $\mathcal{H}$ are constructed by considering the different combinations on the way we connect supernodes corresponding to the endpoints of the edges in G.

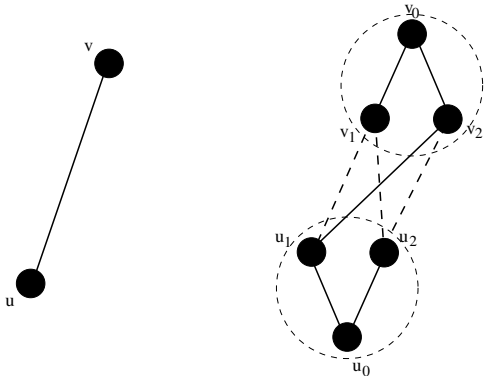


Fig. 2. The construction of graphs in $\mathcal{H}$. The figure shows two supernodes (the dotted circles) corresponding to two adjacent nodes u, v of G. There is an edge between nodes u_1 and v_2; the three dashed edges denote the alternative ways to connect the supernodes.

Consider the nodes u_0 and v_0 of the graphs in $\mathcal{H}$ that correspond to two adjacent nodes u and v of G. By our construction, in every graph of $\mathcal{H}$, the distance between u_0 and v_0 is 3. Any other path connecting these two nodes has length at least $g+1$; to see this, observe that, since G has girth g, any path that connects some of the nodes u_1 and u_2 with some of the nodes v_1 and v_2 (in any graph of $\mathcal{H}$) has length at least $g-1$. So, in any graph of $\mathcal{H}$, the greedy path from node u_0 to node v_0 defined by any embedding with stretch smaller than $\frac{g+1}{3}$ should be identical to the corresponding (unique) shortest path. This is the only requirement we will utilize in our proof; clearly, there are additional necessary requirements which we will simply ignore.

For an ordered pair of adjacent nodes u and v in G, consider the three nodes u_1, u_2, and v_0 of the graphs in $\mathcal{H}$. Let x, y, z be points of $\mathbb{R}^d$ corresponding to these nodes. We will view the coordinates of x, y, and z as real variables. What an embedding f has to do is simply to set the values of these variables, possibly selecting different values for different graphs of family $\mathcal{H}$. Define the quantity $\text{dist}(x,z)^p - \text{dist}(y,z)^p$ and observe that, due to the definition of the distance ℓ_p, this is a polynomial of degree p over the coordinates of x, y, and z. Let us call this polynomial $\mathcal{P}^{u_0 \to v_0}$. Clearly, the sign of $\mathcal{P}^{u_0 \to v_0}$ indicates the result of the comparison of $\text{dist}(x,z)$ with $\text{dist}(y,z)$. Let f be an embedding of stretch smaller than $\frac{g+1}{3}$ and observe that the greedy path from u_0 to v_0 defined by f is identical to the corresponding shortest path only if

- $\mathcal{P}^{u_0 \to v_0}$ has sign -1 for all graphs in $\mathcal{H}$ that connect supernodes u and v through an edge with endpoint at u_1 and
- $\mathcal{P}^{u_0 \to v_0}$ has sign 1 for all graphs in $\mathcal{H}$ that connect supernodes u and v through an edge with endpoint at u_2.

Denote by $\mathcal{P}$ the set of all polynomials $\mathcal{P}^{u_0 \to v_0}$ for every ordered pair of adjacent nodes u and v in G and recall that the graphs of $\mathcal{H}$ are produced by selecting all possible combinations of ways to connect supernodes corresponding to adjacent nodes of G. By repeating the above argument, we obtain that f has stretch smaller than $\frac{g+1}{3}$ for every graph in $\mathcal{H}$ only if the set of polynomials $\mathcal{P}$ realizes all the 4^m different sign patterns.

Here is where we will use a result of Warren [17] which upper-bounds the number of different sign patterns of a set of polynomials.

Theorem 3 (Warren [17]). *Let $\mathcal{P}$ be a set of K polynomials of degree δ on ℓ real variables. Then, the number of different sign patterns $\mathcal{P}$ may have is at most $\left(\frac{4e\delta K}{\ell}\right)^{\ell}$.*

Observe that $\mathcal{P}$ consists of $2m$ polynomials of degree p over $3dN$ real variables. Using Theorem 3, we have that the number of different sign patterns the polynomials of $\mathcal{P}$ can realize is $\#\text{sp} \leq \left(\frac{8emp}{3dN}\right)^{3dN}$.

We will now assume that $d \leq \frac{m}{N \log 8ep}$ in order to show that $\#\text{sp} < 4^m$ and obtain a contradiction. Observe that the bound we obtained using Warren's theorem can be expressed as a function $f(d) = \left(\frac{A}{dB}\right)^{dB}$ where $A = 8emp$ and $B = 3N$. Also, observe that f is increasing in $[0, \frac{A}{eB}]$, i.e., for $d \leq \frac{8mp}{3N}$. Clearly, our assumption satisfies this inequality. Hence, we have

$$\#\text{sp} = \left(\frac{8emp}{3dN}\right)^{3dN} \leq (8ep \log 8ep)^{\frac{m}{\log 8ep}} < (8ep)^{\frac{2m}{\log 8ep}} = 4^m.$$

We have obtained a contradiction. Hence, $d > \frac{m}{N \log 8ep}$. In order to obtain a lower bound on d only in terms of the number of nodes and p, it suffices to use a girth-g graph G that is as dense as possible. The following well-known result indicates that dense girth-g graphs do exist.

Theorem 4 (Erdös and Sachs [7]). *For every $g \geq 3$ and for infinitely many values of N, there are N-node graphs of girth at least g with at least $\frac{1}{4}N^{1+1/g}$ edges.*

We remark that the parameters of this theorem can be slightly improved. For example, the complete N-node graph has girth 3.

Hence, by selecting G to have $m \geq \frac{1}{4}N^{1+1/g}$ edges and recalling that $n = 3N$, we obtain that $d > \frac{N^{1/g}}{4 \log 8ep} \geq \frac{n^{1/g}}{12 \log 8ep}$. The next statement summarizes the discussion in this section.

Theorem 5. *Let $k \geq 3$ be an integer and let f be a greedy embedding of n-node graphs to the metric space $\mathbb{R}^d$ with distance function ℓ_p. If f has stretch smaller than $\frac{k+1}{3}$, then $d > \frac{n^{1/k}}{12 \log 8ep}$.*

An immediate corollary of this statement (for $k \in O\left(\frac{\log n}{\log \log n}\right)$) is that, in general, the best stretch we can achieve using Euclidean spaces with polylogarithmic

number of dimensions and the distance ℓ_2 is $\Omega\left(\frac{\log n}{\log\log n}\right)$. Also, note that using the N-node complete graph (of girth 3) as G, we obtain a lower bound of $\Omega(n/\log p)$ on the number of dimensions for no-stretch greedy embeddings that use the distance ℓ_p. For $p = 2$, we essentially obtain the same bound we have obtained in Section 2.

4 A Lower Bound for the Embedding of Flury et al.

We will now adapt the lower bound of the previous section for the greedy embeddings presented by Flury et al. [8]. The embedding of [8] uses the Euclidean space $\mathbb{R}^d$ and the distance function min-max$_c$ defined as follows. Let c be a factor of d and let $(s_1, s_2, ..., s_d)$ and $(t_1, t_2, ..., t_d)$ be points of $\mathbb{R}^d$. For each $j \in [d/c]$, let

$$D_j = \max_{(c-1)j+1 \le i \le cj} |s_i - t_i|.$$

Then, the function min-max$_c$ is defined as

$$\text{min-max}_c(s,t) = \min_{j \in [d/c]} D_j.$$

We remark that our arguments below can be extended (without any modification) to any distance function such that dist(s,t) is computed by applying operators min and max on the quantities $|s_i - t_i|$ for $i \in [d]$. For example, the ℓ_∞ distance belongs in this category.

In the previous section, the sign of a single polynomial indicated the result of the comparison between two distances. This was possible because of the definition of the ℓ_p distance function. With distance function min-max$_c$, this is clearly impossible. However, the argument of the previous section can be adapted using the sign pattern of a set of polynomials as an indication of the result of the comparison between two distances.

Again, we use the same construction we used in Section 3. Starting from a girth-g N-node graph G with m edges, we construct the family of graphs $\mathcal{H}$. Our argument will exploit the same minimal requirement we exploited in the previous section. Namely, for every ordered pair of adjacent nodes u and v in G, in any graph of $\mathcal{H}$ the greedy path from node u_0 to node v_0 defined by any embedding with stretch smaller than $\frac{g+1}{3}$ should be identical to the corresponding shortest path.

For an ordered pair of adjacent nodes u and v in G, consider the three nodes u_1, u_2, and v_0 of the graphs in $\mathcal{H}$. Let x, y, z be points of $\mathbb{R}^d$ corresponding to these nodes. Again, we view the coordinates of x, y, and z as variables; recall that what an embedding has to do is simply to set the values of these variables. Let us now define the following set of polynomials $\mathcal{Q}^{u_0 \to v_0}$ over the coordinates of x, y, and z:

- The $d(d-1)/2$ polynomials $(x_i - z_i)^2 - (x_j - z_j)^2$ for $1 \le i < j \le d$.
- The $d(d-1)/2$ polynomials $(y_i - z_i)^2 - (y_j - z_j)^2$ for $1 \le i < j \le d$.
- The d^2 polynomials $(x_i - z_i)^2 - (y_j - z_j)^2$ for $i, j \in [d]$.

Notice that there is no way to restrict all the above polynomials to have strictly positive or strictly negative values. So, the term sign in the following corresponds to the outcome of the signum function sgn0 : $\mathbb{R} \to \{-1, 0, +1\}$. Note that the sign of each polynomial above does not change if we replace the squares by absolute values. Hence, for all assignments for which the sign pattern of $\mathcal{Q}^{u_0 \to v_0}$ is the same, the relative order of the absolute values of the difference at the same coordinate between the pairs of points x, z and y, z is identical. Now assume that two embeddings f and f' are such that $\text{min-max}_c(f(u_2), f(v_0)) > \text{min-max}_c(f(u_1), f(v_0))$ and $\text{min-max}_c(f'(u_2), f'(v_0)) < \text{min-max}_c(f'(u_1), f'(v_0))$. This means that the embeddings f and f' correspond to two assignments of values to the coordinates of x, y, and z so that the set of polynomials $\mathcal{Q}^{u_0 \to v_0}$ has two different sign patterns.

Let $\mathcal{Q}$ be the union of the $2m$ sets of polynomials $\mathcal{Q}^{u_0 \to v_0}$ and $\mathcal{Q}^{v_0 \to u_0}$ for every adjacent pair of nodes u, v in G. The above fact implies that in order to find greedy embeddings of stretch smaller than $\frac{g+1}{3}$ for every graph in $\mathcal{H}$, the set of polynomials $\mathcal{Q}$ should have at least 4^m different sign patterns. Again, we will show that this is not possible unless d is large. Since signs are defined to take values in $\{-1, 0, 1\}$ now, we cannot use Warren's theorem in order to upper-bound the number of sign patterns of $\mathcal{Q}$. We will use a slightly different version that applies to our case that is due to Alon [1].

Theorem 6 (Alon [1]). *Let $\mathcal{Q}$ be a set of K polynomials of degree δ on ℓ real variables. Then, the number of different sign patterns $\mathcal{Q}$ may have (including zeros) is at most $\left(\frac{8e\delta K}{\ell}\right)^{\ell}$.*

Observe that $\mathcal{Q}$ contains at most $2d^2m$ degree-2 polynomials in total and the number of variables is $3dN$. By applying Theorem 6, the different sign patterns the polynomials of $\mathcal{Q}$ may have are $\#\text{sp} \leq \left(\frac{64emd}{3N}\right)^{3dN}$.

We will now assume that $d \leq \frac{m}{3N \log N}$ in order to obtain a contradiction. Observe that the bound we obtained using Theorem 6 is increasing in d. Hence, we have

$$\#\text{sp} \leq \left(\frac{64emd}{3N}\right)^{3dN} \leq \left(\frac{64em^2}{9N^2 \log N}\right)^{\frac{m}{3 \log N}} < \left(\frac{16eN^2}{9 \log N}\right)^{\frac{m}{\log N}} \leq n^{\frac{2m}{\log N}} = 4^m,$$

where the second inequality follows since $m < N^2/2$ and the third one follows since N is sufficiently large (i.e., $N > 2^{16e/9}$). Again, we have obtained a contradiction. Hence, $d > \frac{m}{3N \log N}$. By setting G to have at least $\frac{1}{4}N^{1+1/g}$ edges (using Theorem 3) and since $n = 3N$, we have $d > \frac{n^{1/g}}{36 \log n}$.

The next statement summarizes the discussion in this section.

Theorem 7. *Let $k \geq 3$ be an integer and let f be a greedy embedding of n-node graphs to the metric space $\mathbb{R}^d$ with distance function* min-max_c*. If f has stretch smaller than $\frac{k+1}{3}$, then $d > \frac{n^{1/k}}{36 \log n}$.*

Again, this statement implies that the best stretch that can be achieved using Euclidean spaces with a polylogarithmic number of dimensions is $\Omega\left(\frac{\log n}{\log \log n}\right)$. Hence, the $O(\log n)$-stretch greedy embedding of [8] is almost best possible.

5 Concluding Remarks

Note that an alternative proof of Theorem 2 could consider the 2^{N^2} different hard crossroads graphs (for all different $N \times N$ sign pattern matrices) and use Warren's theorem and similar reasoning with that we used in Section 3 in order to prove that greedy embeddings in $\mathbb{R}^d$ using distance ℓ_2 that have stretch smaller than 2 require $d \in \Omega(n)$. However, we find it interesting that the same result (and with slightly better constants) follow by adapting the original argument of Maymounkov [13]. Finally, we remark that even though we have focused on Euclidean spaces, our work has implications to embeddings in multi-dimensional hyperbolic spaces as well. We plan to discuss this issue in the final version of the paper.

Acknowledgement. We thank Tasos Sidiropoulos for helpful discussions.

References

1. Alon, N.: Tools from higher algebra. In: Graham, R.L., Grötschel, M., Lovász, L. (eds.) Handbook of Combinatorics, vol. 2, pp. 287–324. MIT Press, Cambridge (1996)
2. Alon, N., Badoiu, M., Demaine, E.D., Farach-Colton, M., Hajiaghayi, M., Sidiropoulos, A.: Ordinal embeddings of minimum relaxation: general properties, trees, and ultrametrics. ACM Transactions on Algorithms 4(4), art. 46 (2008)
3. Alon, N., Frankl, P., Rödl, V.: Geometrical realization of set systems and probabilistic communication complexity. In: Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS), pp. 277–280 (1985)
4. Awerbuch, B., Peleg, D.: Sparse partitions. In: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 503–513 (1990)
5. Boguna, M., Papadopoulos, F., Krioukov, D.: Sustaining the internet with hyperbolic mapping. Nature Communications 1, art. 62 (2010)
6. Eppstein, D., Goodrich, M.T.: Succinct greedy geometric routing using hyperbolic geometry. IEEE Transactions on Computers 60(11), 1571–1580 (2011)
7. Erdös, P., Sachs, H.: Reguläre Graphen gegebener Taillenweite mit minimaler Knotenzahl. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe 12, 251–257 (1963)
8. Flury, R., Pemmaraju, S.V., Wattenhofer, R.: Greedy routing with bounded stretch. In: Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM), pp. 1737–1745 (2009)
9. Forster, J.: A linear lower bound on the unbounded error probabilistic communication complexity. Journal of Computer and System Sciences 65(4), 612–625 (2002)
10. Kleinberg, R.: Geographic routing using hyperbolic space. In: Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM), pp. 1902–1909 (2007)
11. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica 15(2), 215–245 (1995)
12. Matoušek, J.: Lectures on Discrete Geometry. Graduate Texts in Mathematics. Springer (2002)

13. Maymounkov, P.: Greedy embeddings, trees, and Euclidean vs. Lobachevsky geometry (2006) (manuscript)
14. Moitra, A., Leighton, T.: Some results on greedy embeddings in metric spaces. Discrete & Computational Geometry 44(3), 686–705 (2010)
15. Papadimitriou, C.H., Ratajczak, D.: On a conjecture related to geometric routing. Theoretical Computer Science 341(1), 3–14 (2005)
16. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. Journal of the ACM 36(3), 510–530 (1989)
17. Warren, H.E.: Lower bounds for approximation by nonlinear manifolds. Transactions of the American Mathematical Society 138(1), 167–178 (1968)

The Fault Tolerant Capacitated k-Center Problem

Shiri Chechik and David Peleg

Department of Computer Science, The Weizmann Institute, Rehovot, Israel
{shiri.chechik,david.peleg}@weizmann.ac.il

Abstract. The *capacitated K-center (CKC)* problem calls for locating K service centers in the vertices of a given weighted graph, and assigning each vertex as a client to one of the centers, where each service center has a limited service capacity and thus may be assigned at most L clients, so as to minimize the maximum distance from a vertex to its assigned service center. This paper studies the fault tolerant version of this problem, where one or more service centers might fail simultaneously. We consider two variants of the problem. The first is the *α-fault-tolerant capacitated K-Center (α-FT-CKC)* problem. In this version, after the failure of some centers, all nodes are allowed to be reassigned to alternate centers. The more conservative version of this problem, hereafter referred to as the *α-fault-tolerant conservative capacitated K-center (α-FT-CCKC)* problem, is similar to the α-FT-CKC problem, except that after the failure of some centers, only the nodes that were assigned to those centers before the failure are allowed to be reassigned to other centers. We present polynomial time algorithms that yields 9-approximation for the α-FT-CKC problem and 17-approximation for the α-FT-CCKC problem.

1 Introduction

Problems and Results. This paper considers the *capacitated K-center (CKC)* problem, where it is required to locate K service centers in a weighted graph, and to assign each of the vertices to one of the service centers, where each service center has a limited service capacity and may be assigned at most L vertices, so as to minimize the maximum distance from a vertex to its assigned service center. All nodes serve as clients, namely, it is required to assign each node to a center even though the node may contain a center place on it.

In the fault tolerant version of this problem, one or more service centers might fail simultaneously. After the failure of the service centers, it is still required to assign each node to some surviving center, obeying the constraint that each center can still serve at most L nodes. The objective is to minimize the maximum distance from a node to its assigned center, under all possible subsets of up to α failed service centers, for some integer α.

We consider two variants of the problem. The first is the *α-fault-tolerant capacitated K-Center (α-FT-CKC)* problem. In this version, after the failure of some centers, all nodes are allowed to be reassigned to other centers. The second variant is a more conservative version of this problem, hereafter referred to

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 13–24, 2012.

as the *α-fault-tolerant conservative capacitated K-center (α-FT-CCKC)* problem, which is similar to the α-FT-CKC problem, except that after the failure of some centers, only the nodes that were assigned to those centers before the failure are allowed to be reassigned to other centers. All other nodes continue to be served by their original centers. We present a polynomial time algorithms that yields 9-approximation for the α-FT-CKC problem and 17-approximation for the α-FT-CCKC problem.

Our definition assumes that a failed node can no longer host a center and supply demands, but its own demands must still be supplied. Notice that in all of the fault-tolerant problems mentioned above, the capacity L must be greater than 1, since if $L = 1$, then all nodes in the graph must be allocated as centers, and if one or more of the nodes fail, then there is not enough overall capacity to handle all nodes.

Related Work. The basic K-center problem is defined as follows. For a given weighted graph G, it is required to select a set S of up to K nodes that will host service centers, and to assign each vertex to a service center, so as to minimize the maximum distance from a vertex to the its assigned service center. Formally, the objective function is $\min_{|S|\leq K} \max_{v\in V}\{\delta_G(v, S)\}$, where $\delta_G(v, S)$ is the distance in G from v to its closest node in S. The problem is known to be NP-hard [5] and admits a 2-approximation algorithm [6,7,8,9,10].

Some generalizations of the K-center problem were considered in the literature (e.g. [2,3,9,13]). One generalization for the K-center problem, referred to as the *capacitated K-center* problem, was introduced by Bar-Ilan, Kortsarz and Peleg [1]. In this version of the problem it is required to locate K service centers in a graph, and to assign each of the vertices to one of the service centers, where each service center has a limited capacity and may be assigned at most L vertices, so as to minimize the maximum distance from a vertex to its assigned service center. An approximation algorithm with ratio 10 was presented in [1] for this problem. This approximation ratio was later improved by Khuller and Sussmann [12] to 6, or to 5 in the version where a single node is allowed to host several service centers.

The basic K-center problem was considered in a failure-prone setting in [11]. In this version it is again required to create K service centers, but some of these centers may fail. After the failure, each node is assigned to the closest surviving center. Here, too, the objective is to minimize the maximum distance from a node to its assigned service center. The problem is parameterized by an integer parameter α, bounding the maximum number of centers that may fail in the worst case. In this version, where each center has an unlimited capacity, the problem can be given the following alternative formulation. Each node is assigned to $\alpha + 1$ service centers and the objective is to minimize the maximum distance from a node to any of its $\alpha + 1$ service centers. Two subversions of this problem were considered in [11]. In the first subversion, every node is required to have $\alpha+1$ centers close to it, whereas in the second subversion, this requirement is applied only to a node that does not have a center placed on it. For the first subversion, a 3-approximation algorithm is given for any α and a 2-approximation algorithm

for $\alpha < 3$. For the second subversion, a 2-approximation ratio algorithm is given for any α. Observe that in the capacitated version of the problem, which is studied here, it is harder to manage α-fault tolerance, since it is not enough to make sure that each node has $\alpha + 1$ service centers close to it; the difficulty is that it could be the case that the nearby service centers do not have enough free capacity to handle this node.

Preliminaries. A solution for the capacitated K-center problem, in both the failure free and failure-prone settings, is a set R of up to K nodes in which centers are to be located. Once the set of locations R is determined, it is then required to assign every node v (as a client) to a service center $ctr(v)$ from R in such a way that the number of nodes served by each center is at most the capacity L. Formally, letting $dom(r)$ denote the set of clients served by a center r, $dom(r) = \{v \mid ctr(v) = r\}$, it is required that $|dom(r)| \leq L$ for every $r \in R$. Such an assignment is termed a *feasible* assignment. A solution R is feasible if it has sufficiently many centers to handle the capacity of all nodes, namely, $K \geq \lceil n/L \rceil + \alpha$. The cost of a solution R is the maximum distance from a node to its assigned center.

Note that in the failure free setting, given a solution R, one can efficiently find an optimal feasible assignment, namely, an assignment for each node to a center of R, that satisfies the constraint that each center serves at most L nodes, and minimizes the cost. This can be done using the following bottleneck method. Sort all edge weights in nondecreasing order, let the sorted list of edges be $e_1, ..., e_{|E|}$. We assume G is a complete weighted graph (a non-complete graph G can be made complete by defining the weight of each edge (x, y) as the length of the shortest path between x and y in G). Note that the cost of every feasible solution R is equal to $\omega(e_i)$ for some $1 \leq i \leq m$. For each weight $W = \omega(e_i)$, define the graph G_W to be the subgraph obtained from G by taking only edges of weight at most W. Consider each possible value of W from $\omega(e_1)$ to $\omega(e_{|E|})$. For each W, the goal is to check if there is a way to assign all nodes to R using only edges from G_W, under the constraint that each center in R can serve at most L nodes. This can be done by defining a suitable flow problem and finding the max flow on it. Construct a bipartite graph $\tilde{G} = (R, V, E')$ where for every $r \in R$ and $v \in V$, there is an edge (r, v) in E' iff the distance from r to v is at most W. Set the capacity of these edges to 1. Add two auxiliary nodes s and t, connect s to each node in R with an edge of capacity L and add an edge of capacity 1 from each node of V to t. If the maximum (s, t)-flow on $\tilde{G}$ is $|V|$, then the K center problem has an assignment of cost W (where a client v is assigned to a center r if the edge (v, r) is used in the flow).

Given a solution R, denote by $\rho(R)$ its cost, namely, the minimum W such that there is a way to assign all nodes to R using only edges from G_W, under the capacity constraint. In the failure-prone setting, given a solution R of the capacitated K-Center problem and a set of failed service centers F, denote by $\rho(R, F)$ the minimum weight W such that there is a way to assign all nodes to centers in $R \setminus F$ using only edges from G_W, under the constraint that each center in $R \setminus F$ can serve at most L clients, namely, $\rho(R, F) = \rho(R \setminus F)$. Denote

the worst case service radius by $\rho_\alpha(R) = \max_{|F|\leq\alpha}\{\rho(R,F)\}$. The α-FT-CKC problem requires finding a solution R that minimizes $\rho_\alpha(R)$. In contrast, in the α-FT-CCKC problem the cost of the optimal solution might be higher than $\rho_\alpha(R)$, due to the additional constraint that only the nodes that were served by the set of failed centers F can be reassigned.

2 High Level Structure of the Algorithm

We consider the *α-fault-tolerant capacitated K-Center (α-FT-CKC)* problem, where the solution should be resilient against a failure of up to α centers, for constant $\alpha \geq 1$. Our algorithms for both versions are based on using the (non-fault-tolerant) algorithms proposed by Khuller and Sussmann [12] as a starting point, and introducing the necessary modifications to make them tolerant against the failure of some centers. For completeness we first describe the main ideas of [12] (which in turn follow [7,1]).

As in [4,7,8,9,12], the algorithms we present follow the general strategy of the bottleneck method. Turn G into a complete graph as described in Section 1, and sort all edge weights in nondecreasing order; let the sorted list of edges be $e_1, ..., e_{|E|}$. For each weight W, define the graph G_W to be the subgraph of G containing only edges e_i of weight $\omega(e_i) \leq W$. Run algorithm $\mathsf{Main}(G_W, K, L)$ for each value W from $\omega(e_1)$ to $\omega(e_{|E|})$, until a feasible solution (with K or fewer centers) is obtained. In each iteration, consider the subgraph G_W and treat it as an unweighted graph. In this graph, define the distance $dist(u, v, G_W)$ between two nodes as the number of edges on the shortest path between them. For the weight W, let K^*_W denote the minimal number of centers needed for a feasible solution of cost at most W. Alg. Main finds a solution for the problem on G_W using some number K_W of centers. We prove that if $K_W > K$, then $K^*_W > K$, i.e., there is no feasible solution of cost at most W using at most K centers.

Note that instead of sequentially iterating on $w(e_i)$ values to find a feasible solution, one can rather invoke a binary search for improving the running time.

For a node $v \in V$, let $\Gamma_i(v) = \{u \mid dist(u, v, G_W) \leq i\}$, and $N_i(v) = \{u \mid dist(u, v, G_W) = i\}$. The algorithms presented in [12] use three central procedures. The first procedure, referred to as Select_Monarchs, first constructs the *power graph* G^2_W, obtained from G_W by adding edges between all pairs that have a common neighbor, and then selects an independent set M in G^2_W and places centers on them. The nodes of M are referred to as *monarchs*.

The set M of monarchs is selected by the following iterative process. Initially, all nodes are unmarked. After choosing a new monarch m, mark all unmarked nodes at distance (in G_W) 1 or 2 from it. The set of new marked nodes is referred to as the *empire* of m, denoted $Emp(m)$. In each iteration we choose a new monarch m' by picking an unmarked node m' that is adjacent to a marked node v. This v is termed the *deputy* of m', and v's monarch, namely, the monarch m such that $v \in Emp(m)$, is termed the *parent* monarch of m' and denoted $Parent(m')$. Note that v must be at distance 2 from its monarch, and in addition, m' is the only monarch at distance 1 from v. For a monarch m, $Emp(m) \subseteq \Gamma_2(m)$

in G_W. This procedure yields a rooted tree T of monarchs with the property that the distance between a monarch and its parent monarch in G_W is three.

A second procedure of [12], referred to as Assign_Domains, tries to assign to each monarch a domain of L nodes at distance at most 2 from it in G_W. A monarch may be assigned vertices from the empires of other monarchs, in case no more nodes from its empire are unassigned.

The third Procedure, ReAssign, handles all unassigned nodes. This procedure creates new centers and assigns all unassigned nodes to centers. The centers are chosen for each connected component separately; this can be done as no nodes will be assigned to centers in different connected components of G_W assuming the optimal cost is W.

The main algorithm is given in Procedure Main. This main procedure is used also for all algorithms presented in this paper. However, the internal components need to be modified for each version of the problem.

Algorithm Main$(G = (V, E), K, L)$

1. for every edge weight W in non-decreasing order of weights do:
   - let $G_W = (V, E_W)$ where $E_W = (e \in E \mid \omega(e) \leq W)$.
   - unmark all vertices.
   - if Assign_Centers(G_W) then exit.

Algorithm Assign_Centers(G_W)

1. Suppose G_W consists of connected components $G_W^1, ..., G_W^\ell$
2. for $1 \leq c \leq \ell$, let n_W^c = number of nodes in connected component G_W^c.
3. let $K_W \leftarrow \sum_c \lceil n_W^c / L \rceil$.
4. if $K_W > K$ then return false.
5. for each connected component G_W^c do:
   - Select_Monarchs(G_W^c)
   - Assign_Domains(G_W^c)
   - ReAssign(G_W^c)
6. if the total number of centers used is more than K then return false.
7. return true.

3 Constant Approximation for the α-FT-CKC Problem

In this section we show a constant approximation algorithm for the α-FT-CKC problem. Recall that in this version of the problem, after the failure of a set of centers F, all nodes can be reassigned again.

In this algorithm, we modify procedures Select_Monarchs, Assign_Domains and ReAssign of [12], and in addition introduce another procedure, named ReAssign_by_F, which is invoked after the failure of a set of centers F. (Formal codes for all procedures, and proofs for all claims, are deferred to the full paper.)

Lemma 1. *If the α-FT-CKC problem has a solution of cost at most $W = w(e_i)$ for some $1 \leq i \leq m$, then the minimum node degree in G_W is at least α. Moreover, every solution of cost at most W requires having $\alpha + 1$ service centers in $\Gamma_1(v)$ for every $v \in V$.*

By Lemma 1, we restrict ourself to the case where the minimum node degree in G_W is at least α. Our Procedure Select_Monarchs first finds a set M_1 of monarchs, which is a maximal independent set in G_W^2, in a way similar to Procedure Select_Monarchs of [12] (as described in Section 2). We call these monarchs *major* monarchs. The algorithm of [12] required a single type of monarchs. In contrast, we introduce a second type of monarchs, referred to as *minor* monarchs, which are selected as follows. For every major monarch m, choose a set of $\alpha - 1$ neighbors of m, not including the deputy of m. (Note that the degree of m is at least α, hence such $\alpha - 1$ neighbors must exist.) Denote this set by $minors(m)$, and set $major(m') = m$ for each $m' \in minors(m) \cup \{m\}$. For every major monarch m, denote the set $\{m\} \cup minors(m)$ by $team(m)$. Let M_2 be the set of all minor monarchs. As claimed above, the optimal solution must contain at least $\alpha + 1$ service centers in $\Gamma_1(m)$, therefore at least α of m's neighbors must host centers. The reason for choosing just $\alpha - 1$ neighbors, instead of α, is that we would like to keep the deputy of m available for placing a center on it in Procedure ReAssign, and setting centers at only $\alpha - 1$ neighbors is sufficient for our needs, as will become clearer later on. In Procedure Select_Monarchs, each time a new major monarch m is selected, all unmarked nodes at distance at most 2 from m are marked and assigned to the empire $Emp(m)$ of m.

We would like to serve as many nodes as possible using the monarchs of $M = M_1 \cup M_2$, where every monarch m can serve clients from $\Gamma_2(major(m))$. Procedure Assign_Domains solves this problem by constructing a suitable graph $\tilde{G}$ and finding the minimum cost maximum flow on it. This procedure constructs a bipartite graph $\tilde{G} = (M, V, E')$, where M is the set of all monarchs, V is the set of nodes of the graph and E' contains an edge (m, v) for every monarch m and node v in $\Gamma_2(major(m))$. The goal now is to assign as many nodes as possible to the service centers, where a node v can be assigned to a center m if it has an edge to it in $\tilde{G}$, i.e., $(v, m) \in E'$. Add to $\tilde{G}$ new nodes s and t and edges $\{(s, m) \mid m \in M\}$ and $\{(v, t) \mid v \in V\}$. For every monarch $m \in M$ set a capacity of $u(s, m) = L$ and for every node $v \in V$ set a capacity of $u(m, v) = 1$. Set the cost of edge (m, v) to be $c(m, v) = 0$ if $m = v$ and 1 otherwise. Compute a min-cost maximum integral flow in this new graph. Set $dom(m)$ to be all nodes that got a unit flow from m. For a set of monarchs $X \subseteq M$, let $dom(X) = \bigcup_{m \in X} dom(m)$. We note that, unlike in [12], here the monarchs can be at distance 1 from one another. Therefore, the monarchs can serve one another, we hence do not assume that all monarchs serve themselves (it could be that by forcing monarchs to serve themselves we will hurt the maximum flow). However, by setting the cost of the edge from a monarch to itself in the graph $\tilde{G}$ to be 0 (and the rest of the edges 1), we make sure that even if a monarch does not serve itself it is still served by another monarch (otherwise we can get a lower cost solution with the same maximum flow). In other words, all monarchs are served.

We now turn to describe Procedure ReAssign. For every major monarch m, let $unassigned(m)$ be the set of all nodes in its empire $Emp(m)$ that are unassigned, namely, that do not belong to the domain of any monarch. Consider the tree T of major monarchs as described in Section 2. The algorithm assigns clients to the

monarchs in a bottom-up process on T. More precisely, the process maintains a copy T' of T consisting of all the monarchs that have not been handled yet. In each step of the algorithm, pick a leaf m from the tree T' for processing and remove it from T'. Let $k'L+\epsilon$ be the number of nodes in $unassigned(m)$ plus the number of unassigned nodes passed to m by its children monarchs in T. Allocate k' new centers at free nodes in m's empire and assign the remaining ϵ unassigned nodes to the monarch m, releasing at most ϵ nodes from m's domain, and pass these nodes to m's parent in T for reassignment (we say that a node is free if no center is placed on it). If m is the root monarch of T, then allocate enough new centers to serve all $k'L + \epsilon$ unassigned nodes. For each of these new allocated centers, we say that m is their major monarch. Notice that if up to α failures might occur, then any feasible solution must contain at least $\lceil n/L \rceil + \alpha$ centers, as otherwise, after the failure of any α centers, there is not enough capacity left to handle all nodes. So in the last step of Procedure ReAssign, check if the algorithm creates fewer than $\lceil n/L \rceil + \alpha$ centers; if so, then add new centers to complete to $\lceil n/L \rceil + \alpha$.

A *light* monarch is one whose domain is of size less than L. Let K_L denote the number of light monarchs and n_L be the number of vertices belonging to the domains of light monarchs, and let n be the total number of vertices.

The following lemmas summarize some basic properties. Define the set of *useful monarchs* U as follows. Let U_0 be the set of light monarchs. Increase U_0 by an iterative process, in which U_j is created by U_{j-1} by adding to it any monarch that contains in its domain a node that could have been assigned to a node in U_{j-1}. Formally, let

$$U_j = U_{j-1} \cup \{m \in M \mid \exists v \in dom(m), \exists m' \in U_{j-1} \text{ and } dist(v, major(m')) \leq 2 \text{ in } G_W\}.$$

Let U be the largest set obtained in this way. We say that a monarch m is *overloaded* if there are unassigned nodes in $\Gamma_2(major(m))$.

We note that the definition of overloaded monarchs is close to the definition of heavy monarchs introduced in [12]. The difference between the two terms is that an overloaded monarch m has unassigned vertices in $\Gamma_2(major(m))$ instead of in its empire. In addition, note that the definition of U is similar to the one presented in [12] with the slight modification that we use $dist(v, major(m'))$ instead of $dist(v, m')$. The reason for these modifications is that in our Procedure Assign_Domains, every monarch m' can serve nodes from $\Gamma_2(major(m'))$ rather than from $\Gamma_2(m')$.

Lemma 2. *(1) The set U does not contain any overloaded monarchs.*
(2) Consider some major monarch $m \in M_1$. If one of the nodes in $team(m)$ belongs to U, then all of them do.

Denote by $dom^*(\theta)$ the set of nodes that are served by some center θ in the optimal solution.

Lemma 3. *(1) Consider a monarch $m \in U \cap M_1$ and a a center θ in an optimal solution such that $\theta \in \Gamma_1(m)$. Then $dom^*(\theta) \subseteq dom(U)$.*

(2) In every solution of radius cost W, the number of centers required satisfies $K^*_W \geq \max\{K_L + \lceil (n - n_L)/L \rceil, \lceil n/L \rceil + \alpha\} = K_W$.
(3) Each major monarch $m \in M_1$ has sufficiently many free nodes in its empire to allocate centers for all the clients in unassigned(m) and passed(m).

We now turn to describe the main difference between the algorithm of [12] and ours, designed to handle also failures. When a set F of at most α centers fails, each of the clients of $dom(F)$ needs to be reassigned to one of the surviving centers. Hence after the failure of a set of centers F, we invoke the procedure ReAssign_by_F, which handles the reassignment, possibly by performing a sequence of transfers until reaching a "free spot", i.e., a surviving center that currently serves fewer than L clients. First note that as we make sure that there are at least $\lceil n/L \rceil + \alpha$ centers, the capacity of all surviving centers is sufficient to serve all nodes. The idea of Procedure ReAssign_by_F is to assign each node $x \in dom(F)$ to a "free spot" by a sequence of transfers that reassign x to some other center, possibly causing another node to become unassigned, then looking for a center for this client, and so on, until reaching the "free spot".

More precisely, Procedure ReAssign_by_F is as follows. Let $F = \{f_1, ..., f_k\}$, for some $k \leq \alpha$, be the set of failed centers. Think of every surviving center r as having $|dom(r)|$ "occupied" capacity and $f(r) = L - |dom(r)|$ "free" places for hosting new clients. Consider all nodes of $dom(F)$ and assign for each unassigned node v a unique free place in a non-faulty center. Let $free(v)$ be that center. Note that at most $f(r)$ nodes can be assigned to the same center r if it contains $f(r)$ free places, namely it serves $L - f(r)$ nodes. The idea now is to perform a sequence of transfers from each unassigned node v until reaching the free place in $free(v)$. For each unassigned node $v \in dom(f)$ for some $f \in F$, find the shortest *monarch path* $MP(v)$ in the tree of monarchs T between the major monarch of f and the major monarch of $free(v)$ (namely, the major monarch m such that $free(v) \in Emp(m)$). Let $MP(v) = (m_1, ..., m_j)$ be this shortest path, where m_1 is the major monarch of f and m_j is the major monarch of $free(v)$. Naively, we could apply the following process. Assign v to m_1, and to enable that, cancel the assignment of some other node v_1 in m_1, making it unassigned. Next, assign v_1 to m_2, making some other node v_2 unassigned, and so on, and by a sequence of transfers reach m_j and make some node v_j that was originally served by m_j unassigned and finally assign v_j to the center $free(v)$.

However, this naive rolling process does not necessarily yield a "good" solution, namely a solution in which each node is assigned to a "relatively" close center. To see this, note that for all unassigned nodes v, the shortest monarch path $MP(v)$ in T from the major monarch of v to the major monarch of $free(v)$ could pass through some major monarch m. As m can serve only L nodes, it cannot serve all nodes passed to it during this process, and it will have to pass some of these nodes further. This could result in a large approximation ratio. Luckily, we can use the minor monarchs in order to avoid such a situation. Recall that each major monarch has $\alpha - 1$ minor monarchs at distance 1 from it. We select, for each failed center f and for each major monarch m, such that $f \notin team(m)$, a different non-faulty center from $team(m)$, and denote this center by $\chi(f, m)$.

Consider now an unassigned node v and let $f \in F$ be the center that serves v previous to the failure event. Let $MP(v) = (m_1, ..., m_j)$ be this shortest path in T from the major monarch of f to the major monarch of $free(v)$. We assign v to $\chi(f, m_2)$, thereby releasing some other node v_2 that was originally served by $\chi(f, m_2)$ and making it unassigned. Repeating this, we get a sequence of transfers that reaches m_j and releases some node v_j that was originally served by $\chi(f, m_j)$. Finally, assign this node v_j to the center $free(v)$.

The following lemma establishes the desired stretch bound. Let W be the first value for which Algorithm Assign_Centers returns true.

Lemma 4. *(1) Let c^* be the optimal solution to the fault-tolerant capacitated K-center problem, namely, the set of vertices such that $\rho_\alpha(c^*)$ is minimal. Then $\rho_\alpha(c^*) \geq W$.*
(2) After the failure of a set F where $|F| \leq \alpha$, the reassignment process ensures that each client is assigned a center at distance at most 9 in G_W.

Corollary 1. *Algorithm* Main *yields an approximation ratio of 9 for the fault-tolerant capacitated K-center problem.*

4 Constant Approximation for the α-FT-CCKC Problem

We now present a constant approximation algorithm to the α-FT-CCKC problem. For the same reasons mentioned in Section 3, we consider only the case where the minimum node degree in G_W is at least α.

Relationship with CKC. The following lemma shows that the α-FT-CCKC problem might requires creating more centers than the CKC problem. We say that a set of nodes $A \subseteq V$ is *k-independent* if every two nodes in A are at distance at least k apart. Let $K^*_{\text{CKC}}(W)$ be the minimal number of centers needed for a feasible solution of cost at most W for the CKC problem. Similarly, let $K^*_{\alpha\text{-FT-CCKC}}(W)$ be the minimal number of centers needed for a feasible solution of cost at most W for the α-FT-CCKC problem. Let $R^*_{\text{CKC}}(W)$ be a solution to the Capacitated K-Centers of cost at most W with minimal number of centers and let $R^*_{\alpha\text{-FT-CCKC}}(W)$ be a solution to the α-FT-CCKC of cost at most W with minimal number of centers.

Lemma 5. *For a 7-independent $A \subseteq V$, $K^*_{\alpha\text{-FT-CCKC}}(W) \geq K^*_{\text{CKC}}(W) + \alpha|A|$.*

The algorithm. The next lemma shows that if the α-FT-CCKC problem admits a feasible solution of cost at most W, namely, $K^*_{\alpha\text{-FT-CCKC}}(W) \leq K$, then for every node v there are sufficiently many nodes in $\Gamma_4(v)$ to allocate α backup centers (that will not serve any node as long as there are no failures) and sufficiently many additional centers to serve all nodes in $\Gamma_4(v)$. If this is not the case, namely, $\Gamma_4(v)$ is too small, then there is no feasible solution of cost at most W, and the algorithm has to proceed to the next possible weight W'. This will be needed in our algorithm when placing centers at the nodes.

Lemma 6. *If the α-FT-CCKC problem admits a feasible solution of cost at most W, then for every node v, the set $\Gamma_4(v)$ has sufficiently many nodes to allocate centers for all nodes in $\Gamma_4(v)$ and for α additional backup servers, i.e., $|\Gamma_4(v)| \geq \alpha + \lceil |\Gamma_4(v)|/L \rceil$.*

As in Section 3, we employ Procedures Select_Monarchs, Assign_Domains and ReAssign, using the same Procedure Assign_Domains as in [12] and modifying Procedures Select_Monarchs and ReAssign.

As explained above, Procedure Select_Monarchs results a tree T of major monarchs. The idea behind assigning the monarchs in Procedure Select_Monarchs is as follows. In Section 3 we could settle for α backup centers, due to the assumption that all nodes can be reassigned after the failure of some centers. In contrast, in the current setting we need to spread many additional backup centers on the graph, in order to ensure that each node v has sufficiently many backup centers close to it. We select major monarchs at some distance from each other, and for each major monarch we allocate α of its neighbors as backup centers. In order to make sure that each major monarch can "afford" to allocate α backup centers and still have enough nearby nodes to allocate as centers to all nodes in its empire and to nodes passed to it, we first make sure that each major monarch m has all the nodes $\Gamma_4(m)$ in its empire. By Lemma 6 this guarantees that each monarch m has enough nodes to allocate centers to α backup centers and to handle all nodes in $\Gamma_4(m)$. To ensure that, we select the major monarchs so that they are at distance at least 10 from each other, and each monarch is at distance exactly 10 from its parent monarch. All nodes in $\Gamma_4(m)$ are assigned to m's empire and nodes at distance 5 from some major monarchs are assigned to the first selected monarch. For a major monarch m, define the deputy of m as some node that is at level-5 of m's parent monarch and its distance to m is 5, where a node v is on level-k of some monarch $\tilde{m}$ if v belongs to $Emp(\tilde{m})$ and v is at distance k from $\tilde{m}$. Note that in contrast to the setting in Section 3, here a node may be the deputy of more than one major monarch. In order to prove that a major monarch m has enough free nodes to allocate centers for all nodes passed to it from its children, we make sure each deputy will get at most $L-1$ nodes from the monarchs it serves as their deputy. For all other unassigned nodes, we allocate centers in the empire of some of these children.

Formally, the major monarchs are selected by an iterative process. Initially set Q to contain an arbitrary node v. While Q has unmarked nodes v such that $dist(v, M_1) \geq 10$ do the following. If $M_1 = \emptyset$ then remove a node v from Q (in this case Q contains only one node) else remove an unmarked node v such that $dist(v, M_1) = 10$ from Q. Make v a major monarch, add it to M_1 and mark it. Add all unmarked nodes in $\Gamma_5(v)$ to $Emp(v)$ and mark them. For each node w in $N_{10}(v)$ (distance 10 from v) such that there exists a node u in $Emp(v) \cap N_5(v) \cap N_5(w)$ do the following. Is w is unmarked and $w \notin Q$ then set v to be w's parent in T, setting $Parent(w) = v$, set the deputy of w to be u, and add w to Q. In addition, for every node $m \in M_1$, choose α arbitrary nodes at distance 1 from m and make them backup centers.

Observe that for every $m \in M_1$, $\Gamma_4(m) \in Emp(m)$. In addition, note that after this process ends, there could be some nodes that are unassigned to any empire, namely, nodes that are at distance more than 5 from all major monarchs and thus were not selected to be in the empire of any major monarch. Observe that these nodes are at distance at most 9 from some major monarchs (as otherwise they would have been selected as major monarchs). The purpose of the second stage of Procedure Select_Monarchs is to handle these unassigned nodes. In this stage we allocate minor monarchs and assign all unassigned nodes to those monarchs. This is again done by an iterative process as follows.

In each iteration, choose an unassigned node m' that is a neighbor of some node u such that $u \in N_5(m) \cap Emp(m)$ for some major or minor monarch m. Make m' a minor monarch, place it in M_2 and assign m' all unassigned nodes at distance 5 from it. Set the parent of m' to be $Parent(m') = m$ and the deputy of m' to be u. In the end of this process, all nodes are assigned to the empire of some monarch and the distance from a minor monarch to its parent is 6. Let M_2 be the set of minor monarchs.

Procedure Assign_Domains is again similar to the one presented in [12]. The procedure assigns as many nodes as possible to all chosen major and minor monarchs, where once again, each monarch can serve all nodes at distance at most 2 from it. The domain $dom(m)$ of a monarch m is the set of all nodes assigned to it by this procedure.

Procedure ReAssign takes care of all nodes that are not served by any center. The main difference with respect to Procedure ReAssign in Section 3 is that here, a node may be the deputy of more than one monarch. We need to make sure each deputy receives at most $L-1$ nodes in total from all the monarchs it serves as deputy. We do that by allocating centers in the empires of some of these children. Formally, let $unassigned(m)$ be the set of all nodes in its empire that are unassigned, namely the nodes in its empire that do not belong to the domain of any monarch. Let T be the tree of monarchs. The process maintains a copy T' of T consisting of all the monarchs that have not been handled yet. In each step of the algorithm, pick a leaf m from the tree T' for processing and remove it from T'. Now for each node u at level-5 of m, let $kL+\epsilon$ be the number of nodes passed to m by its children monarchs in T such that u serves as their deputy. If $k > 0$ then allocate k new centers in the empires of the monarchs that u serves as their deputy. Assign kL from $passed(u)$ to those new centers. Pass the ϵ remaining nodes to m. The next step takes care of all unassigned nodes in m's empire and the nodes passed to m by its children in the tree T. Let $k'L+\epsilon$ be the number of nodes in m's empire that are unassigned plus the number of nodes passed to m. Allocate k' new centers in the empire of monarch m. Allocate the ϵ remaining nodes at m possibly displacing ϵ other nodes from m's original clients. Add the displaced nodes to the list of Passed nodes of the deputy of m unless m is the root and then allocate a new center at m's empire and assign the unassigned nodes to it.

Analysis.

Lemma 7. $K^*_{\alpha\text{-FT-CCKC}}(W) \geq K_L + \lceil (n - n_L)/L \rceil + \alpha|M_1| = K_W$.

The following lemma shows that each time the algorithm has to allocate new centers, there are sufficiently many free nodes to do so.

Lemma 8. *(1) Each monarch m has sufficiently many available nodes in its vicinity to allocate centers for $passed(m)$ and $unassigned(m)$ nodes.*
(2) Let u be the deputy of a set of monarchs S and assume $|passed(u)| = kL + \epsilon$ for some integer $k \geq 0$. Then there are at least k available nodes to allocate centers in the empires of S.

Finally, the following lemma establishes the desired stretch bound.

Lemma 9. *Under all possible subsets of up to α failed service centers, each vertex v is assigned to a center w s.t. $dist(v, w) \leq 17$ in G_W.*

Large capacities and Multi k-centers. We note that in the special case where $\alpha < L$, we can improve the approximation ratio to 13. In addition, in the simpler variant of this problem, where a node may host several centers, using Lemma 5 we can establish a 6-approximation ratio for this problem. Details are deferred to the full paper.

References

1. Bar-Ilan, J., Kortsarz, G., Peleg, D.: How to allocate network centers. J. Algorithms 15, 385–415 (1993)
2. Bar-Ilan, J., Kortsarz, G., Peleg, D.: Generalized Submodular Cover Problems and Applications. Theoretical Computer Science 250, 179–200 (2001)
3. Dyer, M., Frieze, A.M.: A simple heuristic for the p-center problem. Oper. Res. Lett. 3, 285–288 (1985)
4. Edmondsa, J., Fulkersona, D.R.: Bottleneck extrema. J. Combinatorial Theory 8, 299–306 (1970)
5. Garey, M.R., Johnson, D.S.: Computers and Intractibility: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1978)
6. Gonzalez, T.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science 38, 293–306 (1985)
7. Hochbaum, D.S., Shmoys, D.B.: Powers of graphs: A powerful approximation algorithm technique for bottleneck problems. In: Proc. 16th ACM Symp. on Theory of Computing, pp. 324–333 (1984)
8. Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Mathematics of Operations Research 10, 180–184 (1985)
9. Hochbaum, D.S., Shmoys, D.B.: A unified approach to approximation algorithms for bottleneck problems. J. ACM 33(3), 533–550 (1986)
10. Hsu, W.L., Nemhauser, G.L.: Easy and hard bottleneck location problems. Discrete Appl. Math. 1, 209–216 (1979)
11. Khuller, S., Pless, R., Sussmann, Y.: Fault tolerant k-center problems. Theoretical Computer Science 242, 237–245 (2000)
12. Khuller, S., Sussmann, Y.: The Capacitated K-Center Problem. SIAM J. Discrete Math. 13, 403–418 (2000)
13. Plesnik, J.: A heuristic for the p-center problem in graphs. Discrete Appl. Math. 17, 263–268 (1987)
14. Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica 2, 385–393 (1982)

Notions of Connectivity in Overlay Networks*

Yuval Emek[1], Pierre Fraigniaud[2], Amos Korman[2],
Shay Kutten[3], and David Peleg[4]

[1] ETH Zurich, Zurich, Switzerland
yuval.emek@tik.ee.ethz.ch
[2] CNRS and University of Paris Diderot, France
(pierref,amos.korman)@liafa.jussieu.fr
[3] The Technion, Haifa, Israel
kutten@ie.technion.ac.il
[4] The Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il

Abstract. "How well connected is the network?" This is one of the most fundamental questions one would ask when facing the challenge of designing a communication network. Three major notions of connectivity have been considered in the literature, but in the context of traditional (single-layer) networks, they turn out to be equivalent. This paper introduces a model for studying the three notions of connectivity in *multi-layer* networks. Using this model, it is easy to demonstrate that in multi-layer networks the three notions may differ dramatically. Unfortunately, in contrast to the single-layer case, where the values of the three connectivity notions can be computed efficiently, it has been recently shown in the context of WDM networks (results that can be easily translated to our model) that the values of two of these notions of connectivity are hard to compute or even approximate in multi-layer networks. The current paper shed some positive light into the multi-layer connectivity topic: we show that the value of the third connectivity notion can be computed in polynomial time and develop an approximation for the construction of well connected overlay networks.

Keywords: Overlay networks, graph theory, connectivity, approximation.

1 Introduction

1.1 Background and Motivation

The term "connectivity" in networks has more than one meaning, but these meanings are equivalent in "traditional" networks. While the graph theoretic definition of connectivity refers to the ability to reach every node from every

* Supported in part by a France-Israel cooperation grant ("Mutli-Computing" project) from the France Ministry of Science and Israel Ministry of Science, by the Technion Cordon center for System Engineering, and by the ISF.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 25–35, 2012.

other node (a.k.a. 1-*connectivity*), the term connectivity is often related also to the *survivability* of a network, namely, the ability to preserve 1-connectivity whenever some links fail.[1] In other words, a network G is said to be *k-connected* if it satisfies the following "connectivity property" (CP):

(CP1) G remains connected whenever up to $k-1$ links are erased from it.

However, there are also other meanings to connectivity. A network G is also said to be *k-connected* if it satisfies the following "connectivity property":

(CP2) There exist k pairwise *link-disjoint paths* from s to t for every two nodes s, t in G.

The equivalence of these two properties [13] enables numerous practical applications. For example, one of the applications of the existence of link-disjoint paths (Property (CP2)) is to ensure survivability (Property (CP1)). Often, a backup (link-disjoint) path is prepared in advance, and the traffic is diverted to the backup path whenever a link on the primary path fails. An example is the backup path protection mechanism in SONET networks (see, e.g., [15]).

A third property capturing connectivity is based on the amount of flow that can be shipped in the network between any source and any destination, defining the capacity of a single link to be 1. In other words, a network G is said to be *k-connected* if it satisfies the following "connectivity property":

(CP3) It is possible to ship k units of flow from s to t for every two nodes s, t in G.

This property too is equivalent to the first two [8], and is also used together with them. For example, routing some flow of information around congestion (which may be possible only if the network satisfies property (CP3) and thus can support this additional flow) uses the second property, i.e., it relies on the existence of multiple link-disjoint paths.

Current networks, however, offer multi-layered structures which yield significant complications when dealing with the notion of connectivity. In particular, the overlay/underlying network dichotomy plays a major role in modeling communication networks, and overlay networks such as peer-to-peer (P2P) networks, MPLS networks, IP/WDM networks, and SDH/SONET-WDM networks, all share the same overall structure: an *overlay* network H is implemented on top of an *underlying* network G. This implementation can be abstracted as a *routing scheme* that maps overlay connections to underlying routes. We comment that there are sometimes differences between such a mapping and the common notion of a routing scheme. Still, since the routing scheme often defines the mapping, we shall term this mapping the *routing scheme*.

Often, the underlying network itself is implemented on top of yet another network, thus introducing a multi-layer hierarchy. Typically, the lower level underlying network is "closer" to the physical wires, whereas the higher level network

[1] The current paper does not deal with *node* connectivity.

is a traffic network in which edges capture various kinds of connections, depending on the context. For the sake of simplicity, we focus on a pair of consecutive layers G and H. This is sufficient to capture a large class of practical scenarios.

The current paper deals with what happens to the different connectivity properties once we turn to the context of overlay networks. As discussed later on, connectivity has been studied previously in the "overlay network" world under the "survivability" interpretation (CP1), and it has been observed that, in this context, the connectivity parameter changes, i.e., the connectivity of the overlay network may be different from that of the underlying network. Lee et al. [12] demonstrated the significance of this difference by showing that the survivability property is computationally hard and even hard to approximate in the multi-layer case. Since the three aforementioned connectivity parameters may differ in multi-layer networks (see Sect. 1.2), they also showed a similar result for the disjoint paths connectivity property.

Interestingly, the motivation of Lee et al. for addressing the disjoint paths connectivity property was the issue of flow. One of the contributions of the current paper is to directly address this issue, showing that in contrast to the previous two notions of connectivity, the maximum flow supported by an overlay network can actually be computed in polynomial time.

In the specific context of survivability, other papers have shown that the issue of connectivity in an overlay network is different from that of connectivity in underlying networking. Consider, for example, a situation where several overlay edges (representing connections) of H pass over the same physical link. Then all these overlay edges may be disconnected as a result of a single hardware fault in that link, possibly disconnecting the overlay network. The affected overlay links are said to *share* the *risk* of the failure of the underlying physical link, hence they are referred to in the literature as a *shared risk link group* (SRLG). An SRLG-based model for overlay networks was extensively studied in recent years;[2] see, e.g., [14] for a useful introduction to this notion and [2] for a discussion of this concept in the context of MPLS. The SRLG model hides the actual structure of the underlying network, in the sense that many different underlying networks can yield the same sets of SRLG. (For certain purposes, this is an advantage of the model.) An even more general notion is that of *Shared Risk Resource Group* However, sometimes this model abstracts away too much information, making certain computational goals (such as, e.g., flow computations) harder to achieve.

In contrast to the SRLG model, we present the alternative model of *deep connectivity*, which allows us to simultaneously consider all three components: the overlay network, the underlying network, and the mapping (the routing scheme). Note that all three should be considered: For example, if the underlying network is not connected, then neither can the overlay network be. The routing scheme also affects the connectivity properties as different routing schemes may yield significantly different overlay link dependencies. In some cases, routing is constrained to shortest paths, whereas in other cases it can be very different.

[2] Note that a common underlying link is not the only possible shared risk; overlay links sharing a node may form a shared risk link group too.

In *policy based* routing schemes (see, e.g. [16]), for example, some underlying edges are not allowed to be used for routing from u to v, which may cause the underlying path implementing the overlay link (u, v) to be much longer than the shortest (u, v)-path.

1.2 The Deep Connectivity Model

The underlying network is modeled by a (simple, undirected, connected) graph G whose vertex set $V(G)$ represents the network nodes, and whose edge set $E(G)$ represents the communication links between them. Some nodes of the underlying network are designated as *peers*; the set of peers is denoted by $\mathcal{P} \subseteq V(G)$. The overlay network, modeled by a graph H, spans the peers, i.e., $V(H) = \mathcal{P}$ and $E(H) \subseteq \mathcal{P} \times \mathcal{P}$; H typically represents a "virtual" network, constructed on top of the peers in the underlying communication network G.

An edge (u, v) in the overlay graph H may not directly correspond to an edge in the underlying graph G (that is, $E(H)$ is not necessarily a subset of $E(G)$). Therefore, communication over a (u, v) edge in H should often be routed along some multi-hop path connecting u and v in G. This is the role of a *routing scheme* $\rho : \mathcal{P} \times \mathcal{P} \to 2^{E(G)}$ that maps each pair (u, v) of peers to some simple path $\rho(u, v)$ connecting u and v in G. A message transmitted over the edge (u, v) in H is physically disseminated along the path $\rho(u, v)$ in G. We then say that (u, v) is *implemented* by $\rho(u, v)$. For the sake of simplicity, the routing scheme ρ is assumed to be symmetric, i.e., $\rho(u, v) = \rho(v, u)$. More involved cases do exist in reality: the routing scheme may be asymmetric, or may map some overlay edge into multiple routes; the simple model given here suffices to show interesting differences between the various connectivity measures.

When a message is routed in H from a peer $s \in \mathcal{P}$ to a non-neighboring peer $t \in \mathcal{P}$ along some multi-hop path $\pi = (x_0, x_1, \ldots, x_k)$ with $x_0 = s$, $x_k = t$, and $(x_i, x_{i+1}) \in E(H)$, it is physically routed in G along the concatenated path $\rho(x_0, x_1)\rho(x_1, x_2) \cdots \rho(x_{k-1}, x_k)$. In some cases, when the overlay graph H is known, it will be convenient to define the routing scheme over the edges of H, rather than over all peer pairs.

The notion of *deep connectivity* grasps the connectivity in the overlay graph H, while taking into account its implementation by the underlying paths in G. Specifically, given two peers $s, t \in \mathcal{P}$, we are interested in three different parameters, each capturing a specific type of connectivity. In order to define these parameters, we extend the domain of ρ from vertex pairs in $\mathcal{P} \times \mathcal{P}$ to collections of such pairs in the natural manner, defining $\rho(F) = \bigcup_{(u,v)\in F} \rho(u, v)$ for every $F \subseteq \mathcal{P} \times \mathcal{P}$. In particular, given an (s, t)-path π in H, $\rho(\pi) = \bigcup_{e\in\pi} \rho(e)$ is the set of underlying edges used in the implementation of the overlay edges along π.

- The *edge-removal deep connectivity* of s and t in H with respect to G and ρ, denoted by $\mathrm{ERDC}_{G,\rho}(s, t, H)$, is defined as the size of the smallest subset $F \subseteq E(G)$ that intersects with $\rho(\pi)$ for every (s, t)-path π in H; namely, the minimum number of underlying edges that should be removed in order to disconnect s from t in the overlay graph.

- The *path-disjoint deep connectivity* of s and t in H with respect to G and ρ, denoted by $\mathrm{PDDC}_{G,\rho}(s,t,H)$, is defined as the size of the largest collection C of (s,t)-paths in H such that $\rho(\pi) \cap \rho(\pi') = \emptyset$ for every $\pi, \pi' \in C$ with $\pi \neq \pi'$, i.e., the maximum number of overlay paths connecting s to t whose underlying implementations are totally independent of each other.
- The *flow deep connectivity* of s and t in H with respect to G and ρ, denoted by $\mathrm{FDC}_{G,\rho}(s,t,H)$, is defined as the maximum amount of flow[3] that can be pushed from s to t in G restricted to the images under ρ of the (s,t)-paths in H, assuming that each edge in $E(G)$ has a unit capacity. Intuitively, if s and t are well connected, then it should be possible to push a large amount of flow between them.

Example: To illustrate the various definitions, consider the underlying network G depicted in Fig. 1(a), and the overlay network H depicted in Fig. 1(b). The routing scheme ρ assigns each of the 6 overlay edges adjacent to the extreme S and T a simple route consisting of the edge itself. For the remaining three overlay edges, the assigned routes are as follows:

$$\begin{aligned}\rho(U_1,U_4) &= (U_1,M_1,M_2,U_2,U_3,U_4)\\ \rho(M_1,M_4) &= (M_1,M_2,D_2,D_3,M_3,M_4)\\ \rho(D_1,D_4) &= (D_1,D_2,D_3,U_3,U_4,D_4)\ .\end{aligned}$$

The route $\rho(M_1,M_4)$ is illustrated by the dashed line in Fig. 1(a). Note that in the original (underlying) network G, the connectivity of the extreme nodes S and T is 3 under all three definitions. In contrast, the values of the three connectivity parameters for the extreme nodes S and T in the overlay network H under the routing scheme ρ are as follows:

- The edge-removal deep connectivity of s and t in H w.r.t. G and ρ is $\mathrm{ERDC}_{G,\rho}(s,t,H) = 2$.
  Indeed, disconnecting the underlying edge (D_2,D_3) plus any edge of the upper underlying route will disconnect S from T.
- The path-disjoint deep connectivity of s and t in H w.r.t. G and ρ is $\mathrm{PDDC}_{G,\rho}(s,t,H) = 1$.
  Indeed, any two of the three overlay routes connecting S and T share an underlying edge.
- The *flow deep connectivity* of s and t in H w.r.t. G and ρ is $\mathrm{FDC}_{G,\rho}(s,t,H) = 3/2$.
  This is obtained by pushing $1/2$ flow unit through each of the three overlay routes.

For each deep connectivity (s,t)-parameter $X_{G,\rho}(s,t,H)$, we define the corresponding *all-pairs* variant $X_{G,\rho}(H) = \min_{s,t\in\mathcal{P}} X_{G,\rho}(s,t,H)$. When G and ρ are clear from the context, we may remove them from the corresponding subscripts.

[3] In the setting of undirected graphs, flow may be interpreted in two different ways depending on whether two flows along the same edge in opposite directions cancel each other or add up. Here, we assume the latter interpretation which seems to be more natural in the context of overlay networks.

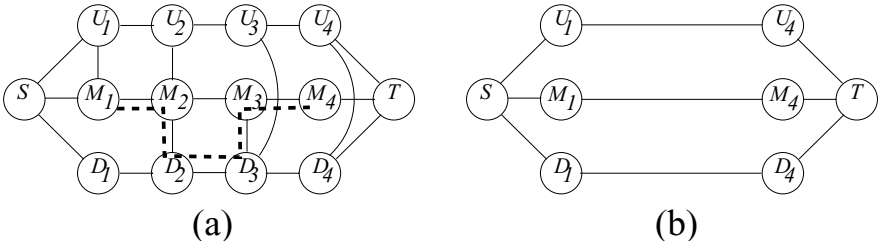


Fig. 1. (a) The underlying graph G. (b) The overlay network H.

1.3 Contribution

Our model for overlay networks makes it convenient to explore the discrepancies between the different deep connectivity notions. Classical results from graph theory, e.g., the fundamental min-cut max-flow theorem [8, 7] and Menger's theorem [13] state that the three connectivity parameters mentioned above are equivalent when a single layer network is considered. Polynomial time algorithms that compute these parameters (in a single layer network) were discovered early [8, 5, 6] and have since become a staple of algorithms textbooks [3, 11]. As mentioned above, previous results [12, 4], when translated to our model, have shown that in multi-layer networks, two of the three connectivity parameters are computationally hard and even hard to approximate. Our first technical contribution is to expand on these negative results by showing that the path-disjoint deep connectivity property cannot be approximated to any finite ratio when attention is restricted to simple paths in the underlying graph.

On the positive side, we show that the flow deep connectivity parameters can be computed in polynomial time (Sect. 3), thus addressing the motivation of [12] for studying the disjoint paths property in overlay networks. Then, we address the issue of constructing a "good" overlay graph for a given underlying graph and routing scheme. As opposed to the difficulty of approximating the value of the parameters, we show that the related construction problem can sometimes be well approximated. Specifically, in Sect. 4, we investigate the problem of constructing 2-edge removal deeply connected overlay graphs with as few as possible (overlay) edges. This problem is shown to be NP-hard, but we show that a logarithmic-approximation for it can be obtained in polynomial-time.

2 Hardness of Approximation

As mentioned earlier, Lee et al. [12] established hardness of approximation results for the problems of computing the parameters $\mathrm{ERDC}_{G,\rho}(s,t,H)$, $\mathrm{PDDC}_{G,\rho}(s,t,H)$, and $\mathrm{ERDC}_{G,\rho}(H)$. For completeness, we observe that the all-pairs variant $\mathrm{PDDC}_{G,\rho}(H)$ of the path-disjoint deep connectivity parameter is also hard to approximate, establishing the following theorem, which is essentially a corollary of Theorem 4 in [12] combined with a result of Håstad [10].

Theorem 1. *Unless NP = ZPP, the problem of computing the parameter* $\mathrm{PDDC}_{G,\rho}(H)$ *cannot be approximated to within a ratio of* $|E(H)|^{1/2-\epsilon}$ *for any* $\epsilon > 0$.

We now turn to show that the following natural variants of the PDDC parameters cannot be approximated to within any finite ratio. Let $\mathrm{SPDDC}_{G,\rho}(s,t,H)$ denote the size of the largest collection C of (s,t)-paths in H such that all paths $\pi \in C$ are implemented by simple paths $\rho(\pi)$ in G and $\rho(\pi) \cap \rho(\pi') = \emptyset$ for every $\pi, \pi' \in C$ with $\pi \neq \pi'$; let $\mathrm{SPDDC}_{G,\rho}(H) = \min_{s,t \in \mathcal{P}} \mathrm{SPDDC}_{G,\rho}(s,t,H)$. Note that these parameters are merely a restriction of the PDDC parameters to simple paths (hence the name, which stands for *simple* path-disjoint deep connectivity).

The inapproximability of the $\mathrm{SPDDC}_{G,\rho}(s,t,H)$ parameter is proved by reducing the set packing problem to the problem of distinguishing between the case $\mathrm{SPDDC}_{G,\rho}(s,t,H) = 0$ and the case $\mathrm{SPDDC}_{G,\rho}(s,t,H) \geq 1$. In fact, the vertices $s,t \in V(H)$ that minimize $\mathrm{SPDDC}_{G,\rho}(s,t,H)$ in this reduction are known in advance, thus establishing the impossibility of approximating the all-pairs parameter $\mathrm{SPDDC}_{G,\rho}(H)$ as well. The proofs of the following two theorems are deferred to the full version.

Theorem 2. *Unless P = NP, the problem of computing the parameter* $\mathrm{SPDDC}_{G,\rho}(s,t,H)$ *cannot be approximated to within any finite ratio.*

Theorem 3. *Unless P = NP, the problem of computing the parameter* $\mathrm{SPDDC}_{G,\rho}(H)$ *cannot be approximated to within any finite ratio.*

3 Efficient Algorithm for FDC

In this section, we develop a polynomial time algorithm that computes the flow deep connectivity parameter $\mathrm{FDC}(s,t,H)$ (which clearly provides an efficient computation of the parameter $\mathrm{FDC}(H)$ as well). Consider some underlying graph G, routing scheme ρ, overlay graph H, and two vertices $s,t \in V(H)$. Let $\mathcal{P}$ denote the collection of all simple (s,t)-paths in H. For each path $p \in \mathcal{P}$ and for each edge $e \in E(G)$, let $\psi(p,e)$ be the number of appearances of the edge e along the image of p under ρ. We begin by representing the parameter $\mathrm{FDC}(s,t,H)$ as the outcome of the following linear program:

$$\begin{array}{ll} \max \sum_{p \in \mathcal{P}} x_p & \text{s.t.} \\ \sum_{p \in \mathcal{P}} \psi(p,e) \cdot x_p \leq 1 & \forall e \in E(G) \\ x_p \geq 0 & \forall p \in \mathcal{P} \end{array}$$

The variable x_p represents the amount of flow pushed along the image under ρ of the path p for every $p \in \mathcal{P}$. The goal is to maximize the total flow pushed along the images of all paths in $\mathcal{P}$ subject to the constraints specifying that the sum of flows pushed through any edge is at most 1. This linear program may exhibit an exponential number of variables, so let us consider its dual program instead:

$$\begin{array}{ll} \min \sum_{e \in E(G)} y_e & \text{s.t.} \\ \sum_{e \in E(G)} \psi(p,e) \cdot y_e \geq 1 & \forall p \in \mathcal{P} \\ y_e \geq 0 & \forall e \in E(G) \end{array}$$

The dual program can be interpreted as fractionally choosing as few as possible edges of G so that the image under ρ of every path p in $\mathcal{P}$ traverses in total at least one edge. We cannot solve the dual program directly as it may have an exponential number of constraints. Fortunately, it admits an efficient separation oracle, hence it can be solved in polynomial time (see, e.g., [9]).

Given some real vector $\boldsymbol{y}$ indexed by the edges in $E(G)$, our separation oracle either returns a constraint which is violated by $\boldsymbol{y}$ or reports that all the constraints are satisfied and $\boldsymbol{y}$ is a feasible solution. Recall that a violated constraint corresponds to some path $p \in \mathcal{P}$ such that $\sum_{e \in E(G)} \psi(p,e) \cdot y_e < 1$. Therefore our goal is to design an efficient algorithm that finds such a path $p \in \mathcal{P}$ if such a path exists.

Let $w(e) = \sum_{e' \in \rho(e)} y_{e'}$ for every edge $e \in E(H)$ and let H' be the weighted graph obtained by assigning weight $w(e)$ to each edge $e \in E(H)$. The key observation in this context is that the (weighted) length of an (s,t)-path p' in H' equals exactly to $\sum_{e \in E(G)} \psi(p,e) \cdot y_e$, where p is the path in H that corresponds to p' in H'. Therefore, our separation oracle is implemented simply by finding a shortest (s,t)-path p^* in H': if the length of p^* is smaller than 1, then p^* corresponds to a violated constraint; otherwise, the length of all (s,t)-paths in H' is at least 1, hence $\boldsymbol{y}$ is a feasible solution. This establishes the following theorem.

Theorem 4. *The parameters* $\mathrm{FDC}_{G,\rho}(s,t,H)$ *and* $\mathrm{FDC}_{G,\rho}(H)$ *can be computed in polynomial time.*

4 Sparsest 2-ERDC Overlay Graphs

In this section, we are interested in the following problem, referred to as the *sparsest* 2-ERDC *overlay graph* problem: given an underlying graph G, a peer set $\mathcal{P} \subseteq V(G)$, and a routing scheme $\rho : \mathcal{P} \times \mathcal{P} \to 2^{E(G)}$, construct the sparsest overlay graph H for $\mathcal{P}$ (in terms of number of overlay edges) satisfying $\mathrm{ERDC}_{G,\rho}(H) \geq 2$. Of course, one has to make sure that such an overlay graph H exists, so in the context of the sparsest 2-ERDC overlay graph problem we always assume that $\mathrm{ERDC}_{G,\rho}(K_{\mathcal{P}}) \geq 2$, where $K_{\mathcal{P}}$ is the complete graph on $\mathcal{P}$. This means that a trivial solution with $\binom{n}{2}$ edges, where $n = |\mathcal{P}|$, always exists and the challenge is to construct a sparser one.

4.1 Hardness

We begin our treatment of this problem with a hardness result.

Theorem 5. *The sparsest* 2-ERDC *overlay graph problem is NP-hard.*

Proof. The assertion is proved by a reduction from the *Hamiltonian path* problem. Consider an n-vertex graph G_0 input to the Hamiltonian path problem. Transform it into an instance of the sparsest 2-ERDC overlay graph problem as follows: Construct the underlying graph G by setting $V(G) = V(G_0) \cup \{x, y\}$

and $E(G) = E(G_0) \cup \{(x,y)\} \cup \{(v,x),(v,y) \mid v \in V(G_0)\}$ and let $\mathcal{P} = V(G_0)$. Define the routing scheme ρ by setting

$$\rho(u,v) = \begin{cases} (u,v) & \text{if } (u,v) \in E(G_0) \\ (u,x,y,v) & \text{otherwise.} \end{cases}$$

This transformation is clearly polynomial in n.

We argue that G_0 admits a Hamiltonian path if and only if there exists an overlay graph H for $\mathcal{P}$ so that $|E(H)| = n$ and $\mathrm{ERDC}_{G,\rho}(H) \geq 2$. To that end, suppose that G_0 admits a Hamiltonian path π. If π can be closed to a Hamiltonian cycle (in G_0), then take H to be this Hamiltonian cycle. Otherwise, take H to be the cycle consisting of π and a virtual edge connecting between π's endpoints. In either case, H clearly has n edges and by the design of ρ, H satisfies $\mathrm{ERDC}_{G,\rho}(H) = 2$.

Conversely, if there exists an overlay graph H for $\mathcal{P}$ so that $|E(H)| = n$ and $\mathrm{ERDC}_{G,\rho}(H) \geq 2$, then H must form a Hamiltonian cycle C in $\mathcal{P} \times \mathcal{P}$. This cycle can contain at most one virtual edge as otherwise, the removal of (x,y) breaks two edges of C which means that $\mathrm{ERDC}_{G,\rho}(H) < 2$. By removing this virtual edge, we are left with a Hamiltonian path in G_0. □

4.2 Constructing Sparse 2-ERDC Overlay Graphs

On the positive side, we develop a polynomial time logarithmic approximation algorithm for the sparsest 2-ERDC overlay graph problem. Our algorithm proceeds in two stages: First, we take T to be an arbitrary spanning tree of $\mathcal{P} \times \mathcal{P}$. Subsequently, we aim towards (approximately) solving the following optimization problem, subsequently referred to as the *overlay augmentation* problem: augment T with a minimum number of $\mathcal{P} \times \mathcal{P}$ edges so that the resulting overlay graph H satisfies $\mathrm{ERDC}_{G,\rho}(H) \geq 2$.

We will soon explain how we cope with this optimization problem, but first let us make the following observation. Denote the edges in $\rho(T)$ by $\rho(T) = \{e_1, \ldots, e_\ell\}$ and consider some overlay graph H such that $E(H) \supseteq T$ and some $1 \leq i \leq \ell$. Let $F_i(H)$ be the collection of connected components of the graph obtained from H by removing all edges $e \in E(H)$ such that $e_i \in \rho(e)$. Fix

$$\kappa_i(H) = |F_i(H)| - 1 \quad \text{and} \quad \kappa(H) = \sum_{i=1}^{\ell} \kappa_i(H)\,.$$

We think of $\kappa(H)$ as a measure of the distance of the overlay graph H from being a feasible solution to the overlay augmentation problem (i.e., $\mathrm{ERDC}(H) \geq 2$).

Proposition 1. *An overlay graph $H \supseteq T$ satisfies $\mathrm{ERDC}(H) \geq 2$ if and only if $\kappa(H) = 0$.*

Proof. If $\mathrm{ERDC}(H) \geq 2$, then $F_i(H)$ must consist of a single connected component for every $1 \leq i \leq \ell$, thus $\kappa(H) = 0$. Conversely, if $\kappa(H) = 0$, then necessarily $|F_i(H)| = 1$ for every $1 \leq i \leq \ell$, which means that H does not disconnect by

the removal of any edge $e_i \in \rho(T)$. It is also clear that the removal of any edge in $E(G) - \rho(T)$ does not disconnect H as the tree T remains intact. Therefore, $\mathrm{ERDC}(H) \geq 2$. □

Consider some edge $e \in (\mathcal{P} \times \mathcal{P}) - E(H)$ and let $H \cup \{e\}$ denote the overlay graph obtained from H by adding the edge e. By the definition of the parameter κ, we know that $\Delta_i(e, H) = \kappa_i(H) - \kappa_i(H \cup \{e\})$ is either 0 or 1 for any $1 \leq i \leq \ell$. Fixing $\Delta(e, H) = \kappa(H) - \kappa(H \cup \{e\})$, we observe that: $\Delta(e, H) = \sum_{i=1}^{\ell} \Delta_i(e, H)$ referred to as Property ($\star$).

We are now ready to complete the description of our approximation algorithm. Starting from $H = T$, the algorithm gradually adds edges to H as long as $\kappa(H) > 0$ according to the following greedy rule. At any intermediate step, we add to H an edge $e \in (\mathcal{P} \times \mathcal{P}) - E(H)$ that yields the maximum $\Delta(e, H)$. When $\kappa(H)$ decreases to zero, the algorithm terminates (recall that this means that $\mathrm{ERDC}(H) \geq 2$).

The analysis of our approximation algorithm relies on the following proposition.

Proposition 2. *The parameter κ can be computed in polynomial time. Moreover, for every two overlay graphs H_1, H_2 such that $E(H_1) \subseteq E(H_2)$, we have*

(1) $\kappa(H_1) \geq \kappa(H_2)$*; and*
(2) $\Delta(e, H_1) \geq \Delta(e, H_2)$ *for every edge $e \in \mathcal{P} \times \mathcal{P}$.*

Proof. The fact that κ can be computed efficiently and the fact that $\kappa(H_1) \geq \kappa(H_2)$ are clear from the definition of κ, so our goal is to prove that $\Delta(e, H_1) \geq \Delta(e, H_2)$ for every $e \in \mathcal{P} \times \mathcal{P}$. By Property ($\star$), it suffices to show that $\Delta_i(e, H_1) \geq \Delta_i(e, H_2)$ for every $1 \leq i \leq \ell$. If $\Delta_i(e, H_2) = 0$, then this holds vacuously, so suppose that $\Delta_i(e, H_2) = 1$. This means that $e_i \notin \rho(e)$ and the endpoints of e belong to different connected components in $F_i(H_2)$. But since $E(H_1) \subseteq E(H_2)$, it follows that the endpoints of e must also belong to different connected components in $F_i(H_1)$, hence $\Delta_i(e, H_1) = 1$ as well. □

Proposition 2 implies that the overlay augmentation problem falls into the class of *submodular cover* problems (cf. [17, 1]) and our greedy approach is guaranteed to have an approximation ratio of at most $\ln(\kappa(T)) + 1 = O(\log N)$, where $N = |V(G)|$. More formally, letting $\hat{H}$ be a sparsest overlay graph such that $\hat{H} \supseteq T$ and $\mathrm{ERDC}(\hat{H}) \geq 2$, it is guaranteed that the overlay graph H generated by our greedy approach satisfies $|E(H) - T| \leq O(\log N) \cdot |E(\hat{H}) - T|$.

To conclude the analysis, let H^* be an optimal solution to the sparsest 2-ERDC overlay graph problem. Clearly, $|E(H^*)| > n - 1 = |T|$. Moreover, since $E(H^*) \cup T$ is a candidate for $\hat{H}$, it follows that $|E(H^*) \cup T| \geq |\hat{H}|$, thus $|E(H^*)| \geq |E(\hat{H}) - T|$. Therefore,

$$\begin{aligned} |E(H)| &\leq O(\log N) \cdot |E(\hat{H}) - T| + |T| \\ &\leq O(\log N) \cdot |E(H^*)| + |E(H^*)| \\ &= O(\log(N) \cdot |E(H^*)|) \end{aligned}$$

which establishes the following theorem.

Theorem 6. *The sparsest* 2-ERDC *overlay graph problem admits a polynomial-time logarithmic-approximation.*

References

[1] Bar-Ilan, J., Kortsarz, G., Peleg, D.: Generalized submodular cover problems and applications. Theor. Comput. Sci. 250(1-2), 179–200 (2001)

[2] Cisco IOS Software Releases 12.0 S. Mpls traffic engineering: Shared risk link groups (srlg), http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/fs29srlg.html

[3] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press (2001)

[4] Coudert, D., Datta, P., Perennes, S., Rivano, H., Voge, M.-E.: Shared risk resource group: Complexity and approximability issues. Parallel Processing Letters 17(2), 169–184 (2007)

[5] Dinic, E.A.: Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. Soviet Math Doklady 11, 1277–1280 (1970)

[6] Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM 19(2), 248–264 (1972)

[7] Elias, P., Feinstein, A., Shannon, C.: A note on the maximum flow through a network. IEEE Transactions on Information Theory 2(4), 117–119 (1956)

[8] Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Canadian Journal of Mathematics 8, 399–404 (1956)

[9] Grotschel, M., Lovasz, L., Schrijver, A.: Geometric algorithms and combinatorial optimization. Combinatorica (1981)

[10] Hastad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. In: IEEE Annual Symposium on Foundations of Computer Science, p. 627 (1996)

[11] Kleinberg, J., Tardos, E.: Algorithm Design. Addison Wesley (2006)

[12] Lee, K., Modiano, E., Lee, H.-W.: Cross-layer survivability in wdm-based networks. IEEE/ACM Transactions on Networking 19(4), 1000–1013 (2011)

[13] Menger, K.: Zur allgemeinen kurventheorie. Fund. Math. 10(95-115), 5 (1927)

[14] Network Protection Website. Network protection techniques, network failure recovery, network failure events, http://www.network-protection.net/shared-risk-link-group-srlg/

[15] Ramamurthy, S., Mukherjee, B.: Survivable wdm mesh networks. part i-protection. In: Proceedings of IEEE Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 1999, vol. 2, pp. 744–751 (March 1999)

[16] White Paper, Cisco Systems Inc. Policy-based routing (1996), http://www.cisco.com/warp/public/cc/pd/iosw/tech/plicy_wp.pdf

[17] Wolsey, L.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica 2, 385–393 (1982) 10.1007/BF02579435

Changing of the Guards: Strip Cover with Duty Cycling

Amotz Bar-Noy[1], Ben Baumer[1], and Dror Rawitz[1,2]

[1] The Graduate Center of the City University of New York
amotz@sci.brooklyn.cuny.edu, bbaumer@gc.cuny.edu
[2] Tel Aviv University
rawitz@eng.tau.ac.il

Abstract. The notion of *duty cycling* is common in problems which seek to maximize the lifetime of a wireless sensor network. In the duty cycling model, sensors are grouped into *shifts* that take turns covering the region in question, and each sensor can belong to at most one shift. We consider the imposition of the duty cycling model upon the Strip Cover problem, where we are given n sensors on a one-dimensional region, and each shift can contain at most $k \leq n$ sensors. We call the problem of finding the optimal set of shifts so as to maximize the length of time that the entire region can be covered by a wireless sensor network, k-Duty Cycle Strip Cover (k-DutySC). In this paper, we present a polynomial-time algorithm for 2-DutySC. Furthermore, we show that this algorithm is a $\frac{35}{24}$-approximation algorithm for k-DutySC. We also give two lower bounds: $\frac{15}{11}$, for $k \geq 4$, and $\frac{6}{5}$, for $k = 3$, and provide experimental evidence suggesting that these lower bounds are tight. Finally, we propose a fault tolerance model and find thresholds on the sensor failure rate, over which our algorithm has the highest expected performance.

1 Introduction

We consider the following problem: Suppose we have a one-dimensional region (or interval) that we wish to cover with a wireless sensor network. We are given the locations of n sensors located on that interval, and each sensor is equipped with an identical battery of finite charge. We have the ability to set the sensing radius of each sensor, but its battery charge drains in inverse proportion to the radius that we set. Our goal is to organize the sensors into disjoint coverage groups (or *shifts*), who will take turns covering the entire region for as long as possible. We call this length of time the *lifetime* of the network.

More specifically, we consider the Strip Cover problem with identical batteries under a *duty cycling* restriction. An instance consists of a set $X \subset [0, 1]$ of n sensor locations, and a rational number B representing the initial battery charge of each sensor. Each battery discharges in inverse linear proportion to its radius, so that a sensor i whose radius is set to r_i survives for B/r_i units of time. In the *duty cycling* model, the sensors are partitioned into disjoint coverage groups, called *shifts*, which take turns covering the entire interval for as long as

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 36–47, 2012.

their batteries allow. The sum of these lengths of time is called the *lifetime* of the network and is denoted by T. For any fixed $k \leq n$, the k-Duty Cycle Strip Cover (k-DutySC) problem seeks an optimal partitioning of the sensors such that the network lifetime T is maximized, yet no coverage group contains more than k sensors. In the fault tolerant variant, each sensor may fail to activate with some fixed probability $p \in [0, 1]$, and we seek to maximize the *expected* lifetime of the network (i.e., the expected sum of lifetimes of surviving shifts).

Solutions to the general Strip Cover problem contain both the radial assignments and activation and de-activation times for each sensors. As a result, these solutions can be complicated to implement and understand. Moreover, interdependence among multiple sensors can make such solutions susceptible to catastrophic decline in network lifetime if there is a non-zero probability of sensor failure. Conversely, since in the duty cycling model each sensor can participate in at most one cover, the scheduling of the covers is of little importance. Furthermore, by minimizing the number of sensors participating in each cover, duty cycling solutions can be insulated from the risk imposed by sensor failure.

Related Work. This line of research began with Buchsbaum, et al.'s [5] study of the Restricted Strip Cover (RSC) problem. In RSC, the locations and sensing radii of n sensors placed on an interval are given, and the problem is to compute an optimal set of activation times, so as to maximize the network lifetime. They showed that RSC is NP-hard, and presented an $O(\log \log n)$-approximation algorithm. Gibson and Varadarajan [11] later improved on this result by discovering a constant factor approximation algorithm.

The problem of finding the optimal set of radial assignments for sensors deployed on an interval, rather than the activation times, is more tractable. Peleg and Lev-Tov [12] considered the problem of covering a finite set of m target points while minimizing the sum of the radii assigned, and found an optimal polynomial-time solution via dynamic programming. The situation wherein the whole interval must be covered corresponds to a "one shift" version of n-DutySC, wherein the restriction is not upon the size of each shift, but upon the number of shifts. Bar-Noy, et al. [4] provided an optimal polynomial-time algorithm for this problem.

The interest in duty cycling developed in part from the introduction of the Set k-Cover problem by Slijepcevic and Potkonjak [16]. This problem, which they showed to be NP-hard, seeks to find at least k disjoint covers among a set of subsets of a base set. Perillo and Heinzelman [15] considered a variation in which each sensor has multiple modes. They translated the problem into a generalized maximum flow graph problem, and employed linear programming to find a optimal solution. Abrams, et al. [1] provided approximation algorithms for a modification of the problem in which the objective was to maximize the total area covered by the sensors. Cardei et al. [6,7,8] considered adjustable range sensors, but also sought to maximize the number of non-disjoint set covers over a set of target coverage points. The work of Pach and Tóth [13,14] also has applications in this context. They showed that a k-fold cover of translates of a centrally-symmetric open convex polygon can be decomposed into $\Omega(\sqrt{k})$

covers. Aloupis, et al. [2] improved this to the optimal $\Omega(k)$ covers, and the centrally-symmetric restriction was later lifted by Gibson and Varadarajan [11]. In each of the above cases, the concept of finding many disjoint set covers, which can be seen as shifts, is used as a proxy for maximizing network lifetime.

Finally, the general Strip Cover problem, in which each sensor has a *different* battery charge, was defined by Bar-Noy, et al. [4]. They did not consider duty cycling, but instead focused on the more general Set Once Strip Cover (OnceSC) problem, in which the radius and activation time of each sensor can be set only once. They showed that OnceSC is NP-hard, and that RoundRobin (sensors take turns covering the entire interval) is a $\frac{3}{2}$-approximation algorithm for both OnceSC and Strip Cover. Bar-Noy and Baumer also analyzed non-duty cycling algorithms for Strip Cover with identical batteries [3]. The Connected Range Assignment problem studied by Chambers, et al. [9], wherein the goal is to connect a series of points in the plane using circles, is also related. They presented approximation bounds for solutions using a fixed number of circles, which is similar to limiting shift sizes.

Our Results. In Section 2, we define the class of k-DutySC problems, and present the trivial solution to 1-DutySC. We present a polynomial-time algorithm, which we call Match, for 2-DutySC in Section 3. In Section 4, we compare the performance of RoundRobin to an algorithm that uses only a single shift. We prove that when the sensors are equi-spaced on the coverage interval, RoundRobin performs most poorly in comparison to the one shift algorithm. Then we study the performance of RoundRobin on these "perfect" deployments. This study is used to analyze Match in k-DutySC, but is of independent interest, since perfect deployments are the most natural. In Section 5 we show that Match is a $\frac{35}{24}$-approximation algorithm for k-DutySC. We also give two lower bounds: $\frac{15}{11}$, for $k \geq 4$, and $\frac{6}{5}$, for $k = 3$, and provide experimental evidence suggesting that these lower bounds are tight. In the full version we consider a fault tolerance model, and show that if the failure rate of each sensor is sufficiently high, Match becomes optimal. We contend that even if the approximation ratio of k-DutySC for $k \geq 3$ is improved, Match will be of interest, due to its simplicity, performance, and fault tolerance.

2 Preliminaries

Duty Cycles. Let $U = [0, 1]$ be the interval that we wish to cover, and let $X = \{x_1, \ldots, x_n\} \in U^n$ be a set of n sensor locations. We assume that $x_i \leq x_{i+1}$, for every $i \in \{1, \ldots, n-1\}$. We first assume that all sensors have unit capacity batteries. We will justify this assumption later.

A pair (C, t), where $C \subseteq X$ is a subset of k sensor locations and $t \geq 0$, is called a *k-duty cycle* (or simply a *duty cycle*, or a *shift*). The sensors in C are activated at the same time and are deactivated together after t time units. A duty cycle (C, t) is feasible if the sensors in C can cover the interval $[0, 1]$ for the duration of t time units. More specifically, a sensor i such that $x_i \in C$ is

assigned a radius $1/t$ and covers the range $[x_i - 1/t, x_i + 1/t]$, and the duty cycle is feasible if $[0,1] \subseteq \bigcup_{i \in C}[x_i - 1/t, x_i + 1/t]$.

Let $\textsc{All}(C)$ denote the maximum t for which (C,t) is feasible. $\textsc{All}(C)$ is called the *lifetime* of C. Given a duty cycle $C = \{x_{i_1}, \ldots, x_{i_k}\}$ define

$$d_j \triangleq \begin{cases} 2x_{i_1} & j = 0, \\ 2(1 - x_{i_k}) & j = k, \\ x_{i_{j+1}} - x_{i_j} & \text{otherwise}, \end{cases} \qquad \text{and} \qquad \Delta \triangleq \max_j \{d_j\} \ .$$

Observation 1 $\textsc{All}(C) = \frac{2}{\Delta}$.

Proof. The maximum lifetime of C is at least $\frac{2}{\Delta}$, since the radial assignment $r_i = \frac{\Delta}{2}$ covers $[0,1]$. However, if $\textsc{All}(C) > \frac{2}{\Delta}$, then $r_i < \frac{\Delta}{2}$, for every i. Hence, $[0,1]$ is not covered. □

In light of Observation 1, it suffices to refer to any subset $C \subseteq X$ as a shift, with a corresponding lifetime that is inferred from $\textsc{All}(C)$.

Problems. k-Duty Cycle Strip Cover (k-DutySC) is defined as follows. The input is a set $X = \{x_1, \ldots, x_n\} \in U^n$ of sensor locations. A solution (or *schedule*) is a partition of X into m non-empty pairwise disjoint subsets $C_1, \ldots, C_m \subseteq X$ such that $|C_j| \leq k$, for every j. The goal is to find a solution that maximizes $\sum_j \textsc{All}(C_j)$. Thus, a solution to DutySC consists of a partition of X into shifts, where each shift employs All to achieve optimal lifetime.

Note that $\textsc{All}(C)$, for any shift C, and hence the maximum lifetime are multiplied by a factor of B, if all sensors have batteries with capacity B. Hence, throughout the paper we assume that all sensors have unit capacity batteries.

The optimal lifetime for k-DutySC is denoted by $\textsc{Opt}_k$. The best possible lifetime of a k-DutySC instance X, for any k, is $2n$.

Observation 2 $\textsc{Opt}_k(X) \leq 2n$, *for every* k.

Proof. Consider a schedule $C_1, \ldots, C_m$ and let Δ_i correspond to C_i. The minimum possible value of Δ_i is $1/|C_i|$, for every i. By Observation 1 we get that $\textsc{All}(C_i) \leq \frac{2}{\Delta_i} \leq 2|C_i|$. The observation follows from the fact that each of the n sensors is used in exactly one shift. □

Perfect Deployment. Define $X_n^* = \left\{\frac{2i-1}{2n} : i \in \{1, \ldots, n\}\right\} = \left\{\frac{1}{2n}, \frac{3}{2n}, \ldots, \frac{2n-1}{2n}\right\}$. We refer to X_n^* as the *perfect deployment* since the n-DutySC lifetime of X_n^* is $2n$, namely $\textsc{All}(X_n^*) = 2n$.

Round Robin. In 1-DutySC each sensor must work alone, therefore there is only one possible solution: $C_i = \{i\}$, for every i. Observe that this solution is valid for k-DutySC, for every k. We refer to the algorithm that generates this solution as RoundRobin. Observe that the RoundRobin lifetime is given by $\textsc{RR}(X) = \sum_i t_i$, where $t_i \triangleq \textsc{All}(C_i) = \max\{1/x_i, 1/(1-x_i)\}$ by Observation 1.

Bar-Noy, et al. [4] showed that RoundRobin is a $\frac{3}{2}$-approximation algorithm for Strip Cover. Since RoundRobin schedules are duty cycle schedules and any k-DutySC schedule is also a Strip Cover schedule, it follows that

Theorem 1. RoundRobin *is a* $\frac{3}{2}$*-approximation algorithm for* k-DutySC*, for every* $k \geq 2$.

The above ratio is tight due to the instance $X_2^* = \left\{\frac{1}{4}, \frac{3}{4}\right\}$ as shown in [3].

3 Strip Cover with Shifts of Size 2

We present a polynomial-time algorithm for solving 2-DutySC. The algorithm is based on a reduction to the Maximum Weight Matching problem in bipartite graphs that can be solved in $O(n^2 \log n + nm)$ in graphs with n vertices and m edges (see, e.g., [10]).

Theorem 2. 2-DutySC *can be solved in polynomial time.*

Proof. Given a 2-DutySC instance X, we construct a bipartite graph $G = (L, R, E)$ as follows: $L = \{v_i : i \in \{1, \ldots, n\}\}$, $R = \{v'_i : i \in \{1, \ldots, n\}\}$, and $E = \left\{(v_i, v'_j) : i \leq j\right\}$. The weight of an edge $e = (v_i, v'_j)$ is defined as $w(e) = \mathrm{RR}(x_i)$, if $i = j$, and $w(e) = \mathrm{All}(\{x_i, x_j\})$, if $i < j$. Observe that a 2-DutySC solution $C_1, \ldots, C_m$ for X induces a (perfect) matching whose weight is the lifetime of the solution. Also, a matching $M \subseteq E$ induces a 2-DutySC solution whose lifetime is the weight of the matching. Hence, the weight of a maximum weight matching in G is the optimal 2-DutySC lifetime of X. □

The algorithm that is described in the theorem is henceforth referred to as Algorithm Match.

4 Round Robin vs. All

Assume we are given a set X of k sensors. In this section we compare $\mathrm{RR}(X)$ to $\mathrm{All}(X)$. This comparison will be used in the next section to analyze Algorithm Match for k-DutySC.

Define $\gamma(X) \triangleq \frac{\mathrm{RR}(X)}{\mathrm{All}(X)}$. We look for a lower bound on $\min_{X:|X|=k} \gamma(X)$. Due to Theorem 1 it follows that $\gamma(X) \geq \frac{2}{3}$, for any set X of k sensors. In what follows, we prove the stronger result that the placement that minimizes the ratio is the perfect deployment, namely X_k^*. Notice that this is true for $k = 2$, since $\gamma(X_2^*) = \frac{2}{3}$. (Using Theorem 1 is not essential, but it simplifies our analysis.)

4.1 Stretching the Instance

Our first step is to transform X into an instance X' for which $\gamma(X') \leq \gamma(X)$ by pushing sensors away from $\frac{1}{2}$ so that all internal gaps are of size Δ. (See Section 2 for the definition of Δ.) If a sensor needs to be moved to the left of 0, it is placed at 0, and if it needs to move to the right of 1, it is placed at 1.

Definition 1. *For a given instance X, let j be the sensor whose location is closest to $\frac{1}{2}$. Then we define the* stretched *instance X' of X as follows:*

$$x'_i = \begin{cases} \max\{0, x_j - (j-i)\Delta\} & i < j, \\ x_j & i = j, \\ \min\{1, x_j + (i-j)\Delta\} & i > j. \end{cases}$$

Lemma 3. *Let X' be the stretched instance of X. Then, $\gamma(X') \leq \gamma(X)$.*

Proof. Sensors only get pushed away from $\frac{1}{2}$, and thus their RoundRobin lifetime only decreases. Thus, $\mathrm{RR}(X') \leq \mathrm{RR}(X)$. By definition, Δ must equal either d_0, d_n or the length of the largest internal gap in X. However neither d_0 nor d_n can be larger in X' than it was in X, since no sensors move closer to $\frac{1}{2}$. Moreover, by construction the length of the largest internal gap in X' is Δ. Hence $\Delta' \leq \Delta$, and $\mathrm{All}(X') \geq \mathrm{All}(X)$. □

4.2 Perfect Deployment Is the Worst

By Lemma 3, it suffices to consider only stretched instances. The next step is to show that the worst stretched instance is in fact the perfect deployment.

Given a stretched instance X' with k sensors, let k_{out} be the number of sensors located on either 0 or 1, and let $k_{in} \triangleq k - k_{out}$ be the number of sensors in $(0,1)$. Notice that $k_{in} \geq 1$. Also notice that if $k = 1$, then $\mathrm{RR}(X') = \mathrm{All}(X')$, and recall that $\gamma(X) \geq \frac{2}{3} = \gamma(X_2^*)$. Therefore we may assume that $k = k_{in} + k_{out} \geq 3$.

Let a and b be the gaps between 0 and the leftmost sensor not at 0, and 1 and the rightmost sensor not at 1, respectively. For reasons of symmetry we assume, w.l.o.g., that $a \leq b$. Hence, $\lceil k_{in}/2 \rceil$ sensors are located in $(0, \frac{1}{2}]$ and $\lfloor k_{in}/2 \rfloor$ sensors are located in $(\frac{1}{2}, 1)$.

The stretched deployment X' can be described as follows:

$$X' = \left\{ 0^{\lfloor k_{out}/2 \rfloor}, a, a + \Delta', \ldots, a + (k_{in} - 1)\Delta' = 1 - b, 1^{\lceil k_{out}/2 \rceil} \right\}$$

Note that $\Delta' = \frac{1-a-b}{k_{in}-1}$, if $k_{in} \geq 2$. Otherwise, if $k_{in} = 1$, then $k_{out} > 1$ and $\Delta' = b$. The RoundRobin lifetime of X' is:

$$\mathrm{RR}(X') = k_{out} + \sum_{i=0}^{\lceil k_{in}/2 \rceil - 1} \frac{1}{1 - (a + i\Delta')} + \sum_{i=0}^{\lfloor k_{in}/2 \rfloor - 1} \frac{1}{1 - (b + i\Delta')} \,. \quad (1)$$

We distinguish three cases:

1. $X' = \{a, a + \Delta', \ldots, a + (k_{in} - 1)\Delta' = 1 - b\}$, where $a \in [0, \Delta'/2]$ and $b \in (0, \Delta'/2]$.

   Let Ω_0 be the set of all such instances. Note that $k_{out} = 0$, if $a > 0$, and that $k_{out} = 1$, if $a = 0$. However, notice that (1) holds if we use $k_{out} = 0$, even for the case where $a = 0$.

2. $X' = \{a, a+\Delta', \ldots, a+(k_{in}-2)\Delta' = 1-b, 1\}$, where $a \in [0, \Delta'/2]$, $b \in [\Delta'/2, \Delta']$.
   Let Ω_1 be the set of all such instances. Note that $k_{out} = 1$, if $a > 0$, and that $k_{out} = 2$, if $a = 0$. However, notice that (11) holds if we use $k_{out} = 1$, even for the case where $a = 0$.
3. $X' = \left\{0^{\lfloor k_{out}/2 \rfloor}, a, a+\Delta', \ldots, a+(k_{in}-1)\Delta' = 1-b, 1^{\lceil k_{out}/2 \rceil}\right\}$, where $a, b \in [0, \Delta']$ and $k_{out} \geq 2$.
   Let $\Omega_{k_{out}}$ be the set of all such instances that correspond to k_{out}. Note that if $a = 0$, (11) holds if we use $k_{out}+1$ in place of k_{out} and $a = \Delta'$.

Lemma 4. *$\gamma(X')$ has no local minima in $\Omega_{k_{out}}$, for any k_{out}.*

Proof. First assume that $k_{in} \geq 2$. Due to (11) and Observation 1 we have that

$$\gamma(X') = \frac{\Delta'}{2}\left[k_{out} + \sum_{i=0}^{\lceil k_{in}/2 \rceil - 1} \frac{1}{1-(a+i\Delta')} + \sum_{i=0}^{\lfloor k_{in}/2 \rfloor - 1} \frac{1}{1-(b+i\Delta')}\right]$$
$$= \frac{k_{out}}{2} \cdot \Delta' + \sum_{i=0}^{\lceil k_{in}/2 \rceil - 1} f_{k_{in}}^{(i)}(a,b) + \sum_{i=0}^{\lfloor k_{in}/2 \rfloor - 1} f_{k_{in}}^{(i)}(b,a) \ ,$$

where $f_{k_{in}}^{(i)}(a,b) = \frac{\Delta'}{2-2(a+i\Delta')}$. Since $\frac{\partial \Delta'}{\partial a} = -\frac{1}{k_{in}-1}$, it follows that

$$\frac{\partial f_{k_{in}}^{(i)}(a,b)}{\partial a} = \frac{(2-2(a+i\Delta'))(\frac{\partial \Delta}{\partial a}) - \Delta(-2-2i\frac{\partial \Delta'}{\partial a})}{(2-2(a+i\Delta'))^2}$$
$$= \frac{2\left(\frac{\partial \Delta'}{\partial a}(1-a)+\Delta'\right)}{4(1-(a+i\Delta'))^2} = \frac{1}{2(k_{in}-1)} \cdot \frac{-b}{(g_{k_{in}}^{(i)}(a,b))^2} \ ,$$

where $g_{k_{in}}^{(i)}(a,b) = 1-(a+i\Delta')$. Thus,

$$\frac{\partial \gamma(X')}{\partial a} = \frac{-k_{out}}{2(k_{in}-1)} + \frac{1}{2(k_{in}-1)}\left[\sum_{i=0}^{\lceil k_{in}/2 \rceil - 1} \frac{-b}{(g_{k_{in}}^{(i)}(a,b))^2} + \sum_{i=0}^{\lfloor k_{in}/2 \rfloor - 1} \frac{-a}{(g_{k_{in}}^{(i)}(b,a))^2}\right] .$$

Since $a = b = k_{out} = 0$ is not possible for any domain $\Omega_{k_{out}}$, we have that $\frac{\partial \gamma(X')}{\partial a} < 0$. Hence $\gamma(X')$ decreases as a increases. An analogous calculation shows that the same is true for $\frac{\partial \gamma(X')}{\partial b}$. Thus, since neither $\frac{\partial \gamma(X')}{\partial a}$ nor $\frac{\partial \gamma(X')}{\partial b}$ can be zero at any point in the interior of the domain $\Omega_{k_{out}}$, $\gamma(X')$ has no local minima. Finally, any minima must occur when both a and b are as large as possible within the domain $\Omega_{k_{out}}$.

It remains to consider the case where $k_{in} = 1$. Since $k_{out} > 1$, we have that $\Delta' = b$. Hence, $\gamma(X') = \frac{b}{2}\left(k_{out} + \frac{1}{b}\right) = \frac{1}{2}(bk_{out}+1)$, which means that $\frac{\partial \gamma(X')}{\partial b} > 0$. Hence, $\gamma(X')$ decreases as b decreases. It follows that the minima occurs when $a = b = \frac{1}{2}$. □

Let $\gamma_k^* = \gamma(X_k^*)$. We show that, for any fixed k, $\gamma(X)$ reaches its minimum at $X = X_k^*$.

Theorem 3. $\min_{X:|X|=k} \gamma(X) = \gamma_k^*$.

Proof. We prove that $\min_{X \in \Omega_{k_{out}}} \gamma(X) \geq \gamma_k^*$ by induction on k_{out}.

For the base case, if $X' \in \Omega_0$, then by Lemma 4, $\gamma(X')$ achieves its minimum on the boundary of Ω_0, when a and b are as large as possible, namely for $a = b = \Delta/2$. In this case, $X' = X_k^*$. Thus, for all $X' \in \Omega_0$, $\gamma(X') > \gamma_k^*$ if $X \neq X_k^*$.

For the inductive step, let $X \in \Omega_{k_{out}}$, for $k_{out} \geq 1$, and assume that $\min_{X \in \Omega_{k_{out}-1}} \gamma(X) \geq \gamma_k^*$. By Lemma 4 it follows that $\gamma(X')$ achieves its minimum on the boundary of $\Omega_{k_{out}}$. If $k_{out} = 1$ (and $k_{in} \geq 2$), then the minimum is when $a = \Delta/2$ and $b = \Delta$, namely for $X' = \{\frac{\Delta}{2}, \frac{3\Delta}{2}, \ldots, 1-\Delta, 1\}$. By symmetry, this instance has the same ratio as the instance $X'' = \{1 - x : x \in X'\}$, which is in Ω_0 with parameters $a = 0$ and $b = \frac{\Delta}{2}$. If $k_{out} > 1$, then the minimum is when $a = \Delta$ and $b = \Delta$. In this case $X' \in \Omega_{k_{out}-1}$ with parameters $a = 0$ and $b = \Delta$. Hence by the induction hypothesis we have that $\gamma(X') \geq \gamma_k^*$. □

4.3 Properties of γ_k^*

We explore γ_k^* as a function of k. Observe that for even k we have that

$$\gamma_k^* = \frac{1}{2k} \cdot 2 \sum_{i=1}^{k/2} \frac{2k}{2k+1-2i} = 2 \sum_{i=1}^{k/2} \frac{1}{2k+1-2i} = 2 \sum_{i=k/2+1}^{k} \frac{1}{2i-1} ,$$

and for odd k we have that

$$\gamma_k^* = \frac{1}{2k} \left[2 + 2 \sum_{i=1}^{(k-1)/2} \frac{2k}{2k+1-2i} \right] = \frac{1}{k} + 2 \sum_{i=(k+1)/2}^{k-1} \frac{1}{2i+1} .$$

Lemma 5. *γ_k^* satisfies the following: (i) $\gamma_k^* \leq \gamma_{k+2}^*$, for every even k. (ii) $\gamma_k^* \geq \gamma_{k+2}^*$, for every odd k. (iii) $\gamma_k^* \geq \gamma_{k+1}^*$, for every odd k.*

Proof. Due to the convexity of the function $f(z) = \frac{1}{z}$, we have that for even k,

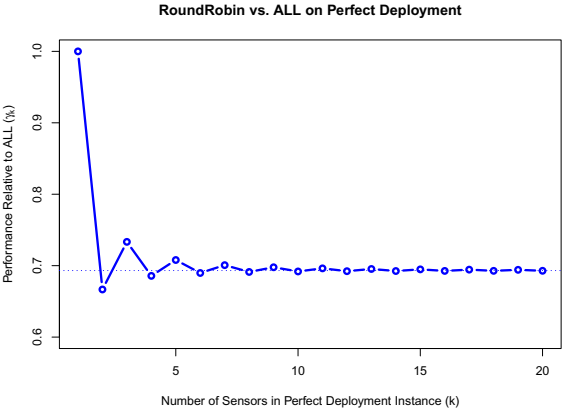
$$\gamma_{k+2}^* - \gamma_k^* = 2 \sum_{i=k/2+2}^{k+2} \frac{1}{2i-1} - 2 \sum_{i=k/2+1}^{k} \frac{1}{2i-1} = \frac{2}{2k+3} + \frac{2}{2k+1} - \frac{2}{k+1} > 0 .$$

and by the same rationale, for odd k,

$$\begin{aligned} \gamma_k^* - \gamma_{k+2}^* &= \left(\frac{1}{k} + 2 \sum_{i=(k+1)/2}^{k-1} \frac{1}{2i+1} \right) - \left(\frac{1}{k+2} + 2 \sum_{i=(k+1)/2+1}^{k+1} \frac{1}{2i+1} \right) \\ &= \frac{1}{k} + \frac{1}{k+2} - \frac{2}{2k+1} - \frac{2}{2k+3} \\ &> 0 . \end{aligned}$$

k	γ_k^*	Approx
1	1	1
2	$\frac{2}{3}$	0.6667
3	$\frac{11}{15}$	0.7333
4	$\frac{24}{35}$	0.6857
5	$\frac{223}{315}$	0.7079
6	$\frac{478}{693}$	0.6898
7	$\frac{6313}{9009}$	0.7007
8	$\frac{4448}{6435}$	0.6912
⋮	⋮	⋮
∞	$\ln 2$	0.6931

(a) Exact and approximate values of γ_k^*.



(b) γ_k^* is an alternating sequence that converges to $\ln 2$

Fig. 1. Tabular and graphical representation of small values of γ_k^*

Finally, for odd k,

$$\gamma_k^* - \gamma_{k+1}^* = \frac{1}{k} + 2\sum_{i=(k+1)/2}^{k-1} \frac{1}{2i+1} - 2\sum_{i=(k+1)/2+1}^{k+1} \frac{1}{2i-1} = \frac{1}{k} - \frac{2}{2k+1} > 0 \ .$$

□

Lemma 6. $\lim_{k\to\infty} \gamma_k^* = \ln 2$.

Proof. Observe that for both even and odd k's we have that $\gamma_k^* \geq \sum_{i=k+1}^{2k} \frac{1}{i} = H_{2k} - H_k$ and $\gamma_k^* \leq \sum_{i=k}^{2k-1} \frac{1}{i} = H_{2k-1} - H_{k-1}$, where H_k the kth Harmonic number. It follows that $\lim_{k\to\infty} \gamma_k^* = \lim_{k\to\infty}(H_{2k} - H_k) = \ln 2$. □

The table in Figure 1(a) contains several values of γ_k^*, whose convergence is also depicted graphically in Figure 1(b).

5 Strip Cover with Shifts of Size k

In this section we analyze the performance of MATCH in k-DUTYSC for $k \geq 3$. Recall that Algorithm MATCH finds the best solution among those using shifts of size at most 2. Since MATCH is more powerful than ROUNDROBIN, its approximation ratio is at most $\frac{3}{2} = 1.5$ (by Theorem 1). We show that the approximation ratio of MATCH is at most $\frac{35}{24} \approx 1.458$. We also provide lower bounds: $\frac{15}{11} \approx 1.364$ for $k \geq 4$, and $\frac{6}{5} = 1.2$, for $k = 3$. At the end of the section we discuss ways to improve the analysis of MATCH.

Upper Bound. We use our analysis of RoundRobin vs. All to obtain an upper bound on the performance of Match for k-DutySC.

Theorem 4. Match *is a* $\frac{35}{24}$*-approximation algorithm for* k-DutySC*, for every* $k \geq 3$.

Proof. Let X be a k-DutySC instance and let $C_1, \ldots, C_m$ be an optimal solution for X. Since for every C_j with $|C_j| \leq 2$, $\textsc{Match}(C_j) = \textsc{Opt}_k(C_j)$, we construct an alternative solution by splitting to singletons every subset C_j such that $|C_j| > 2$. For any such C_j, by Theorem 3 we know that $\textsc{RR}(C_j) \geq \gamma^*_{C_j} \textsc{All}(C_j)$, and by Lemma 5 we know that $\gamma^*_{C_j} \geq \min_{k \geq 3} \gamma^*_k = \gamma^*_4 = \frac{24}{35}$. The lemma follows. □

We can generalize this approach to find upper bounds on the performance of $\textsc{Opt}_k$ in n-DutySC, for $k \leq n$.

Lemma 7. $\textsc{Opt}_k(X) \geq \gamma^*_\ell \textsc{Opt}_n(X)$*, where* ℓ *is the smallest even integer larger than* k.

Hence, the approximation ratio of an algorithm that solves k-DutySC is at most $1/\gamma^*_\ell$. Lemma 6 implies that the least upper bound achievable via this technique is $1/\ln 2 \approx 1.4427$.

Lower Bounds. We show that the approximation ratio of Match is at least $\frac{15}{11}$, for $k \geq 4$, and at least $\frac{6}{5}$, for $k = 3$. The proofs are omitted for lack of space.

Lemma 8. $\textsc{Match}(X^*_4) = \frac{11}{15}\textsc{Opt}_k(X^*_4)$*, for every* $k \geq 4$*, and* $\textsc{Match}(X^*_3) = \frac{5}{6}\textsc{Opt}_k(X^*_3)$.

We conjecture that Lemma 8 is tight.

For some positive integer d, let $D_d = \{\frac{i}{d} : i \in \{0, 1, \ldots, d\}\}$ be a discretization of $[0, 1]$. Clearly, as $d \to \infty$, D becomes a close approximation of $[0, 1]$. Using brute force, we checked all 680 possible instances in D^3_{16} and all 2380 possible instances in D^4_{16}, and found no instance X for which $\textsc{Match}(X) < \frac{5}{6}\textsc{Opt}_3(X)$ in the first case, nor any for which $\textsc{Match}(X) < \frac{11}{15}\textsc{Opt}_4(X)$ in the second case.

Again, due to Lemma 5, we can generalize Lemma 8 to find lower bounds on the performance of $\textsc{Opt}_k$ in n-DutySC.

Lemma 9. *For every* $\ell \leq n$, $\textsc{Opt}_k(X) \geq \frac{\textsc{Opt}_k(X^*_\ell)}{2\ell} \cdot \textsc{Opt}_n(X)$.

Note that For $k = 1, 2$ we recover the $\frac{2}{3}$ and $\frac{11}{15}$ bounds demonstrated previously.

Asymptotics. One way to improve the analysis of Algorithm Match would be to first prove that perfect deployments are worst with respect to Match (as they are with respect to RoundRobin), and then to analyze $\gamma^2_k = \textsc{Match}(X^*_k)/\textsc{All}(X^*_k)$.

Our experiments show that γ^2_k seem to converge to approximately 0.816, which is significantly higher than $\lim_{k\to\infty} \gamma^*_k = \ln 2 \approx 0.693$. See Figure 2.

We would like to evaluate $\lim_{k\to\infty} \gamma^*_2$. However, since Match explicitly evaluates each pair of sensors, it is not a simple algorithm. Nevertheless, we obtain

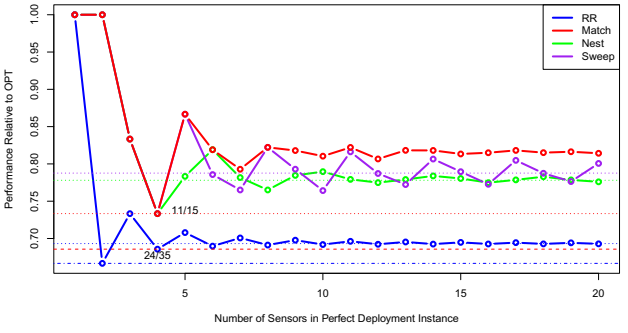


Fig. 2. Performance of RoundRobin, Match, Sweep, and Nest compared to All on perfect deployments

lower bounds on the limit by analyzing simple heuristics. For a given instance X, let $L = X \cap [0, \frac{1}{3}]$, $M = X \cap (\frac{1}{3}, \frac{2}{3})$, and $R = X \cap [\frac{2}{3}, 1]$. Note that $|L| = |R|$ for the perfect deployment instance X_k^* of any size. Consider the following heuristics:

- Sweep: Sensors in L and R are paired to create shifts of size two, starting with the leftmost sensor in L and the *leftmost* sensor in R. Any remaining sensors are put in size one shifts.
- Nest: Sensors in L and R are paired to create shifts of size two, starting with the leftmost sensor in L and the *rightmost* sensor in R. Any remaining sensors are put in size one shifts.

Note that these heuristics return valid (albeit suboptimal) 2-DutySC solutions.

Using similar arguments as in Section 4.3 we can show that:

$$\frac{\text{Sweep}(X_n^*)}{\text{All}(X_n^*)} \approx 1/2 + H_{4n/3} - H_n \underset{n\to\infty}{\to} 0.788$$

$$\frac{\text{Nest}(X_n^*)}{\text{All}(X_n^*)} \approx (H_n - H_{n/2})/2 + (H_{2k/3} - H_{k/2})/2 + (H_{4n/3} - H_n) \underset{n\to\infty}{\to} 0.778$$

A comparison of the performance of both Sweep and Nest on perfect deployments to the performance of All is given in Figure 2.

6 Discussion and Open Problems

While 1-DutySC can be solved trivially by RoundRobin, and we have shown that 2-DutySC can be solved in polynomial time using Algorithm Match, it remains open whether k-DutySC is NP-hard, for $k \geq 3$. It would also be interesting to close the gap between the upper and lower bounds on the approximation ratio of Match, for $k \geq 3$. We offered one possible direction to improving the upper bound in Section 5.

In this paper, we have assumed that: (i) the initial battery charge of each sensor is identical; and (ii) the battery charge in each sensor drains in inverse

linear proportion to its assigned radius. Two natural extensions of our work would be to allow the initial battery charges to differ, and to allow the latter proportion to vary according to some exponent $\alpha \neq 1$.

Finally, while we have restricted our attention to a one-dimensional coverage region, one could consider a variety of similar problems in higher dimensions. For example, one might keep the sensor locations restricted to the line, but consider a two-dimensional coverage region. Conversely, the sensors could be located in the plane, while the coverage region remains one-dimensional. Of course, an even more general problem would allow both the sensor locations and the coverage region to be two-dimensional.

References

1. Abrams, Z., Goel, A., Plotkin, S.A.: Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. In: IPSN, pp. 424–432 (2004)
2. Aloupis, G., Cardinal, J., Collette, S., Langerman, S., Orden, D., Ramos, P.: Decomposition of multiple coverings into more parts. Discrete & Computational Geometry 44(3), 706–723 (2010)
3. Bar-Noy, A., Baumer, B.: Maximizing Network Lifetime on the Line with Adjustable Sensing Ranges. In: Erlebach, T., Nikoletseas, S., Orponen, P. (eds.) ALGOSENSORS 2011. LNCS, vol. 7111, pp. 28–41. Springer, Heidelberg (2012)
4. Bar-Noy, A., Baumer, B., Rawitz, D.: Set It and Forget It: Approximating the Set Once Strip Cover Problem. Arxiv preprint arXiv:1204.1082v1 (2012)
5. Buchsbaum, A., Efrat, A., Jain, S., Venkatasubramanian, S., Yi, K.: Restricted strip covering and the sensor cover problem. In: SODA, pp. 1056–1063 (2007)
6. Cardei, M., Du, D.: Improving wireless sensor network lifetime through power aware organization. Wireless Networks 11(3), 333–340 (2005)
7. Cardei, M., Wu, J., Lu, M.: Improving network lifetime using sensors with adjustable sensing ranges. Int. J. Sensor Networks 1(1/2), 41–49 (2006)
8. Cardei, M., Wu, J., Lu, M., Pervaiz, M.: Maximum network lifetime in wireless sensor networks with adjustable sensing ranges. In: WiMob (3), pp. 438–445 (2005)
9. Chambers, E.W., Fekete, S.P., Hoffmann, H.-F., Marinakis, D., Mitchell, J.S.B., Srinivasan, V., Stege, U., Whitesides, S.: Connecting a Set of Circles with Minimum Sum of Radii. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 183–194. Springer, Heidelberg (2011)
10. Cook, W.J., Cunningham, W.H., William, R., Pulleyblank, A.S.: Combinatorial Optimization. John Wiley & Sons (1998)
11. Gibson, M., Varadarajan, K.: Decomposing coverings and the planar sensor cover problem. In: FOCS, pp. 159–168. IEEE (2009)
12. Lev-Tov, N., Peleg, D.: Polynomial time approximation schemes for base station coverage with minimum total radii. Computer Networks 47(4), 489–501 (2005)
13. Pach, J.: Covering the plane with convex polygons. Discrete & Computational Geometry 1(1), 73–81 (1986)
14. Pach, J., Tóth, G.: Decomposition of multiple coverings into many parts. Computational Geometry 42(2), 127–133 (2009)
15. Perillo, M., Heinzelman, W.: Optimal sensor management under energy and reliability constraints. In: Wireless Communications and Networking, vol. 3, pp. 1621–1626. IEEE (2003)
16. Slijepcevic, S., Potkonjak, M.: Power efficient organization of wireless sensor networks. In: IEEE Intl. Conf. on Communications, vol. 2, pp. 472–476 (2001)

Deterministic Local Algorithms, Unique Identifiers, and Fractional Graph Colouring

Henning Hasemann[1], Juho Hirvonen[2], Joel Rybicki[2], and Jukka Suomela[2]

[1] Institute of Operating Systems and Computer Networks,
TU Braunschweig, Germany
[2] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Finland

Abstract. We show that for any $\alpha > 1$ there exists a deterministic distributed algorithm that finds a fractional graph colouring of length at most $\alpha(\Delta + 1)$ in any graph in one synchronous communication round; here Δ is the maximum degree of the graph. The result is near-tight, as there are graphs in which the optimal solution has length $\Delta + 1$.

The result is, of course, too good to be true. The usual definitions of scheduling problems (fractional graph colouring, fractional domatic partition, etc.) in a distributed setting leave a loophole that can be exploited in the design of distributed algorithms: the size of the local output is not bounded. Our algorithm produces an output that seems to be perfectly good by the usual standards but it is impractical, as the schedule of each node consists of a very large number of short periods of activity.

More generally, the algorithm shows that when we study distributed algorithms for scheduling problems, we can choose virtually any trade-off between the following three parameters: T, the running time of the algorithm, ℓ, the length of the schedule, and κ, the maximum number of periods of activity for a any single node. Here ℓ is the objective function of the optimisation problem, while κ captures the "subjective" quality of the solution. If we study, for example, bounded-degree graphs, we can trivially keep T and κ constant, at the cost of a large ℓ, or we can keep κ and ℓ constant, at the cost of a large T. Our algorithm shows that yet another trade-off is possible: we can keep T and ℓ constant at the cost of a large κ.

1 Introduction

In the study of *deterministic distributed algorithms*, it is commonly assumed that there are *unique numerical identifiers* available in the network: in an n-node network, each node is labelled with a unique $O(\log n)$-bit number.

In the general case, numerical identifiers are, of course, very helpful—many fast distributed algorithms crucially depend on the existence of numerical identifiers, so that they can use the Cole–Vishkin technique [2] and similar tricks. However, when we move towards the fastest possible distributed algorithms, the landscape looks very different.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 48–60, 2012.

1.1 Local Algorithms and Numerical Identifiers

We focus on *local algorithms* [8,11], i.e., distributed algorithms that run in constant time (a constant number of communication rounds), independently of the size of the network. In this context, it is no longer obvious if unique identifiers are of any use:

1. In their seminal work, Naor and Stockmeyer [8] prove that there is a class of problems—so-called LCL problems—that do not benefit from unique numerical identifiers: if an LCL problem can be solved with a local algorithm, it can also be solved with an *order-invariant* local algorithm. Order-invariant algorithms do not exploit the numerical value of the identifier; they merely compare the identifiers with each other and use the relative order of the identifiers.
2. More recently, Göös et al. [3] have shown that for a large class of optimisation problems—so-called PO-checkable problems—local algorithms do not benefit from any kind of identifiers: if a PO-checkable optimisation problem can be approximated with a local algorithm, the same approximation factor can be achieved in anonymous networks if we are provided with a port-numbering and an orientation.

While the precise definitions of LCL problems and PO-checkable problems are not important here, they both share the following seemingly technical requirement: it is assumed that the *size of a local output is bounded by a constant*. That is, for each node in the network, there is only a constant number of possible local outputs, independently of the size of the network. However, previously it has not been known whether this is a necessary condition or merely a proof artefact—while contrived counter-examples exist, natural counter-examples have been lacking.

1.2 Contributions

In this work we provide the missing piece of the puzzle: we show that the condition is necessary, even if we focus on natural graph problems and natural encodings of local outputs. More precisely, we show that there is a classical graph problem—namely, *fractional graph colouring* (see Sect. 2)—with the following properties:

1. In a natural problem formulation, the local outputs can be arbitrarily large.
2. The problem can be solved with a deterministic local algorithm; the algorithm exploits both numerical identifiers and unbounded local outputs.
3. The problem cannot be solved with a deterministic local algorithm without numerical identifiers.
4. The problem cannot be solved with a deterministic local algorithm if we require that the local outputs are of a constant size.

Moreover, this is not an isolated example. The same holds for many other scheduling problems—for example, *fractional domatic partitions* have similar properties.

It is up to the reader's personal taste whether this work should be interpreted as a novel technique for the design of local algorithms, or as a cautionary example of a loophole that needs to be closed.

1.3 Comparison with Other Graph Problems

In the study of local algorithms, we often have to focus on bounded-degree graphs [4,5,6,7]. If we have a constant maximum degree Δ, then a constant-size local output is a very natural property that is shared by a wide range of combinatorial graph problems—at least if we use a natural encoding of the solution:

1. Independent set, vertex cover, dominating set, connected dominating sets, etc.: The output is a subset $X \subseteq V$ of nodes. Each node outputs 1 or 0, indicating whether it is part of X.
2. Matching, edge cover, edge dominating set, spanning subgraphs, etc.: The output is a subset $Y \subseteq E$ of edges. A node of degree d outputs a binary vector of length d, with one bit for each incident edge.
3. Vertex colouring, domatic partition, minimum cut, maximum cut, etc.: The output is a partitioning of nodes, $X_1 \cup X_2 \cup \cdots \cup X_k = V$. Each node outputs an integer $i \in \{1, 2, \ldots, k\}$, indicating that it belongs to subset X_i. In most cases, there is a natural constant upper bound on k: for example, a vertex colouring does not need more than $\Delta + 1$ colours, a domatic partition cannot contain more than $\Delta + 1$ disjoint dominating sets, and a cut by definition has $k = 2$.
4. Graph properties: Each node outputs 1 or 0. For a yes-instance, all nodes have to output 1, and for a no-instance, at least one node has to output 0.

Now if we consider the linear programming (LP) relaxations of problems such as independent sets, vertex covers, or dominating sets, we arrive at a graph problem in which local outputs could be potentially arbitrarily large: each node outputs a rational number, and there is no a priori reason to require that the size of the output (i.e., the length of the binary encoding of the rational number) is bounded. However, it seems that for these problems the size of the output cannot be exploited by a local algorithm—for example, in the case of packing and covering LPs, an exact solution cannot be found by any local algorithm, and the local approximation schemes [6,7] do not need to exploit unbounded local outputs. Indeed, if we had an algorithm that produces arbitrarily large outputs, we could apply a simple rounding scheme without losing too much in the approximation ratio.

However, fractional graph colouring—the LP relaxation of the vertex colouring problem—is a different story. There we not only have unbounded local outputs, but we show that we can exploit this property in the design of local algorithms.

2 Fractional Graph Colouring

In the fractional graph colouring problem, the task is to coordinate the activities of the nodes in a conflict-free manner. Each node has to perform at least one unit of work, and whenever a node is active all of its neighbours have to be inactive. The objective is to minimise the total length of the schedule, i.e., complete the activities as quickly as possible. The applications include the coordination of radio transmissions in a wireless network: each node must transmit one unit of data, and the transmissions of adjacent nodes interfere with each other.

Definitions. Let $G = (V, E)$ be a simple, undirected graph that represents a distributed system: each node $v \in V$ is a computational entity, and each edge $\{u, v\} \in E$ represents a communication link between a pair of nodes. Let

$$\mathcal{I} = \{I \subseteq V : \text{if } u, v \in I \text{ then } \{u, v\} \notin E\}$$

consist of all *independent sets* of G. A *fractional graph colouring* associates a value $x(I) \geq 0$ to each $I \in \mathcal{I}$ such that

$$\sum_{I \in \mathcal{I}:\, v \in I} x(I) \geq 1 \quad \text{for all} \quad v \in V.$$

The *length* of a colouring x is

$$\ell(x) = \sum_{I \in \mathcal{I}} x(I),$$

and an optimal fractional graph colouring minimises $\ell(x)$. See Fig. 1a for an illustration.

The connection between a colouring x and a conflict-free schedule is straightforward: we simply allocate a time slot of length $x(I)$ to I. For example, if we are given a colouring x, we can choose an arbitrary ordering $\mathcal{I} = \{I_1, I_2, \dots\}$ on $\mathcal{I}$, and schedule the activities of the nodes as follows: first all nodes in I_1 are active for $x(I_1)$ time units, then all nodes in I_2 are active for $x(I_2)$ time units, etc.; after $\ell(x)$ time units each nodes have been active for at least one time unit. Conversely, if we can coordinate the activities, we can construct a graph colouring x, as at each point in time the set of active nodes is in $\mathcal{I}$.

Schedules of Nodes. When we study fractional graph colouring in a distributed setting, we assume that each node produces its own part of the solution. That is, each node must know when it is supposed to be active. Formally, the *schedule of a node* $v \in V$ is a union of disjoint intervals

$$s(v) = (a_1, b_1] \cup (a_2, b_2] \cup \cdots \cup (a_k, b_k].$$

Here $0 \leq a_1 < b_1 < a_2 < b_2 < \cdots < a_k < b_k$ are rational numbers. We require that the total length of the time intervals is at least 1, that is, $\sum_i (b_i - a_i) \geq 1$. The *local output* of node v is a binary encoding of the sequence $a_1, b_1, a_2, b_2, \dots, a_k, b_k$.

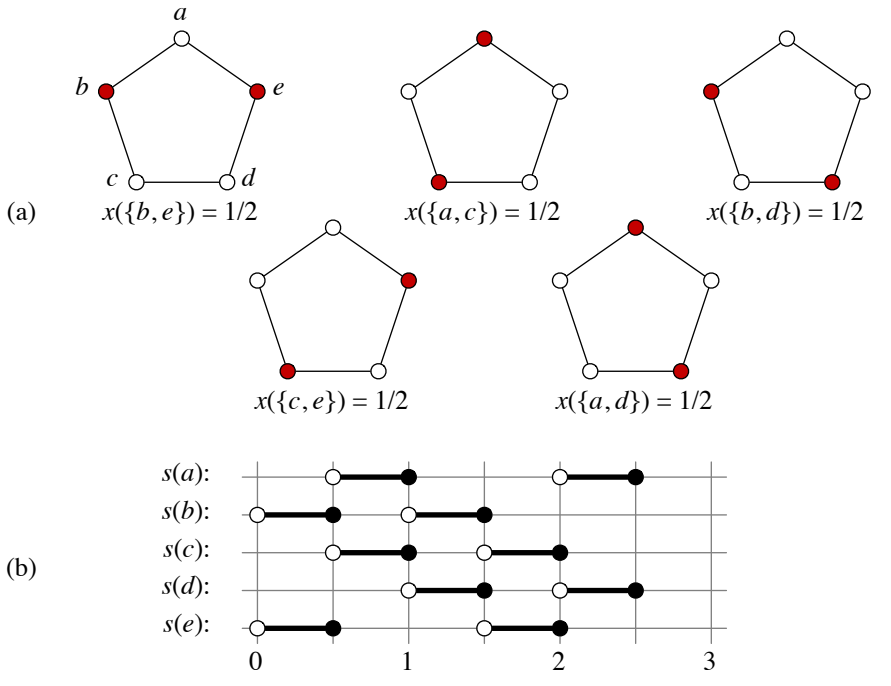


Fig. 1. (a) A fractional graph colouring x of length $\ell(x) = 5/2$ for the 5-cycle. (b) The schedules of the nodes; each node is active for 1 time unit in total, and no node is active after time 5/2.

We say that node v is active at time t if $t \in s(v)$; let $A(t, s) = \{v \in V : t \in s(v)\}$ consist of the nodes that are active at time t. It is straightforward to see that a schedule s defines a fractional graph colouring x of length at most L if

$$A(t, s) = \emptyset \text{ for all } t > L, \quad \text{and} \quad A(t, s) \in \mathcal{I} \text{ for all } t \leq L.$$

Equivalently, we have the locally checkable conditions

$$\max s(v) \leq L \text{ for each } v \in V, \quad \text{and} \quad s(u) \cap s(v) = \emptyset \text{ for each } \{u, v\} \in E.$$

See Fig. 1b for an illustration.

3 Model of Distributed Computing

All of our results hold in the *LOCAL model* [9]. In this model, we assume that each node $v \in V$ has a unique identifier $f(v) \in \{1, 2, \ldots, \mathsf{poly}(|V|)\}$. Initially, each node knows its own identifier. Computation proceeds in synchronous communication rounds. In each round, each node in parallel (1) sends a message to each of its neighbours, (2) receives a message from each of its neighbours, (3) updates its own state. After each round, a node can stop and announce its local

output. All state transitions are deterministic; there is no source of randomness available. The *running time* is the number of communication rounds until all nodes have stopped. The size of a message is unbounded, and we do not restrict local computation.

To keep our positive result as general as possible, we will not use the assumption that we have globally unique identifiers. We only assume that we have some labelling $f\colon V \to \mathbb{N}$ such that $f(u) \neq f(v)$ for each edge $\{u, v\} \in E$. Put otherwise, we only assume that we are given some proper vertex colouring f of G—this is not to be confused with the fractional graph colouring x that we are going to output.

4 Main Results

Now we are ready to give the main result of this work.

Theorem 1. *For any $\alpha > 1$ there exists a deterministic local algorithm $\mathcal{A}$ such that in any graph G algorithm $\mathcal{A}$ finds a fractional graph colouring x for G in one communication round. Moreover, the length of x is at most $\alpha(\Delta + 1)$, where Δ is the maximum degree of G.*

We emphasise that algorithm $\mathcal{A}$ does not need to know the number of nodes in G, the maximum degree of G, or any other properties of G. Moreover, the running time is 1, independently of G. However, the theorem heavily abuses the fact that the size of the output is unbounded—the size of a local output depends on graph G and its labelling f.

The result is near-tight in the sense that there are graphs that do not have a fractional graph colouring of length shorter than $\Delta + 1$. A simple example is the complete graph on $\Delta + 1$ nodes: an optimal fractional graph colouring has length $\Delta + 1$.

From the perspective of the approximability of minimum-length fractional graph colouring, we cannot do much better, either; the following lower bound leaves only a logarithmic gap. Note that the lower bound holds even in the case of d-regular graphs, and even if the running time of the algorithm is allowed to depend on d.

Theorem 2. *Let $\mathcal{F}_d$ be the family of d-regular graphs, and let $\mathcal{A}_d$ be a deterministic algorithm that finds a fractional graph colouring for any $G \in \mathcal{F}_d$ in T_d communication rounds. Then for each d there is a graph $G_d \in \mathcal{F}_d$ such that G_d admits a fractional graph colouring of length 2, but $\mathcal{A}_d$ outputs a fractional graph colouring of length $\Omega(d/\log d)$.*

Incidentally, in the case of triangle-free graphs, the gap could be closed—we could improve the upper bound by borrowing ideas from Shearer's algorithm [10]. Closing the gap for the case of general graphs is left for future work.

5 Proof of Theorem 1

Informally, our algorithm builds on the following idea: We take an appropriate *randomised* algorithm $\mathcal{A}'$ that produces independent sets. The running time of the randomised algorithm is 1, and it does not require that the random numbers are independent for nodes that are not adjacent. Then we build a deterministic schedule that, essentially, goes through a (very large) number of "random" numbers, and feeds these numbers to $\mathcal{A}'$. Then we simply put together all "random" independent sets that are produced by $\mathcal{A}'$. The approach is general, in the sense that we could plug in any randomised algorithm $\mathcal{A}'$ that satisfies certain technical properties. However, to keep the presentation readable, we hard-code a specific concrete choice of $\mathcal{A}'$.

5.1 Preliminaries

Choose $\varepsilon > 0$ and $\beta > 0$ such that

$$\frac{1+\beta}{1-\varepsilon} \leq \alpha.$$

Define $R(x) = \lceil (x+1)/\varepsilon \rceil$. We use the notation $N(v) = \{u \in V : \{u, v\} \in E\}$ for the set of neighbours of $v \in V$, and we write $\deg(v) = |N(v)|$ for the degree of v. Let $N^+(v) = \{v\} \cup N(v)$. The case of an isolated node is trivial; hence we assume that $\deg(v) \geq 1$ for every node v.

5.2 Communication

Recall the definitions of Sect. 3; we assume that we are given a function f that is a proper vertex colouring of graph $G = (V, E)$. The communication part of the algorithm is nearly trivial: *each node v sends its colour $f(v)$ and its degree* $\deg(v)$ *to each of its neighbours.*

This information turns out to be sufficient to find a fractional graph colouring. The rest of this section explains the local computations that are done by each node; they do not involve any communication at all.

5.3 Scheduling Colours

Let $g\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. We say that g is a *scheduling colour function* if

$$\begin{aligned} &g(i,j) \geq j && \text{for all } i \text{ and } j, \\ &g(i,j) \neq g(i',j') && \text{for all } i,\ i',\ j, \text{ and } j' \text{ such that } i \neq i'. \end{aligned}$$

In the algorithm, we will need a scheduling colour function g. For the sake of concreteness, we give an example of such a function:

$$g(i,j) = B(i+j-1) + i - 1, \quad \text{where} \quad B(k) = 2^{\lceil \log_2 k \rceil}.$$

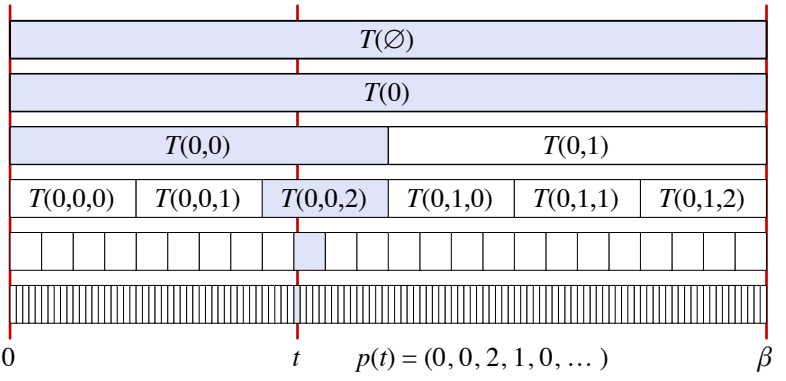


Fig. 2. Recursive partitioning $T(p)$ of the interval $(0, \beta]$

Other choices of g are equally good for our purposes; the choice of g only affects the size of the local outputs.

We define that the *scheduling colour* of a node v is

$$c(v) = g\big(f(v), R(\deg(v))\big).$$

We make the following observations:

1. Function $c\colon V \to \mathbb{N}$ is a proper colouring of G, as f was a proper colouring of G.
2. We have $c(v) \geq R(\deg(v))$ for each node v.
3. Each node v knows $c(u)$ for all $u \in N^+(v)$.

5.4 Coordinates

A *coordinate* is a sequence $p = (p_1, p_2, \ldots, p_\ell)$ where $p_i \in \{0, 1, \ldots, i-1\}$. Here ℓ is the *dimension* of the coordinate; we write $\emptyset$ for the coordinate of dimension $\ell = 0$.

Define $\beta_i = \beta/(i!)$ for each $i \geq 0$. With each coordinate p of dimension ℓ, we associate a time interval $T(p)$ of length β_ℓ as follows (see Fig. 2 for an illustration):

1. For the 0-dimensional coordinate, set $T(\emptyset) = (0, \beta_0]$.
2. Assume that p is a coordinate of dimension $i-1$ with $T(p) = (a, a + \beta_{i-1}]$. For each $j = 0, 1, \ldots, i-1$, we define

$$T(p, j) = \big(a + j\beta_i,\ a + (j+1)\beta_i\big].$$

5.5 First Fragment of the Schedule

Now we are ready to define the schedule within time interval $T(\emptyset)$. To this end, consider a point in time $t \in T(\emptyset)$. Time t defines a unique infinite sequence

$$p(t) = \big(p(1,t), p(2,t), \dots\big)$$

such that for any i we have

$$t \in T\big(p(1,t), p(2,t), \dots, p(i,t)\big).$$

Define the *weight* of the colour class $k \in \mathbb{N}$ at time t as follows:

$$W(k,t) = \frac{p(k,t)}{k}.$$

We define the weight of a node v at time t as the weight of its scheduling colour:

$$w(v,t) = W(c(v),t).$$

Finally, we define that v is active at time t if it is strictly heavier than any neighbour, that is

$$w(v,t) > w(u,t) \quad \text{for all } u \in N(v). \tag{1}$$

Note that each node v knows $c(u)$ for each $u \in N^+(v)$. Hence each node knows when it is active. Moreover, the schedule can be efficiently computed and it is of finite length. To see this, let

$$c'(v) = \max_{u \in N^+(v)} c(u).$$

Let p be a coordinate of length $c'(v)$. Now the weights $w(u,t)$ for $u \in N^+(v)$ are constant during $t \in T(p)$; hence v is either active or inactive during the entire time period $T(c'(v))$. Hence it is sufficient to consider a finite number of time periods.

We will now argue that the schedule for $T(\emptyset)$ is feasible and, moreover, each node is active for a substantial fraction of $T(\emptyset)$. To this end, define

$$h(v) = \frac{1-\varepsilon}{\deg(v)+1}.$$

Lemma 1. *If $\{u,v\} \in E$, nodes u and v are never active simultaneously during $T(\emptyset)$.*

Proof. This is trivial, as we had a strict inequality in (1).

Lemma 2. *Each node $v \in V$ is active for at least $\beta h(v)$ time units within time interval $T(\emptyset)$.*

Proof. Assume that we choose a point in time $t \in T(\emptyset)$ uniformly at random. Then the random variables $p(i,t) \in \{0,1,\ldots,i-1\}$ for $i = 1,2,\ldots$ are independent and uniformly distributed; it follows that the random variables $W(i,t)$ are also independent and uniformly distributed. For any i and any $0 \le x \le 1$ we have

$$\Pr\bigl[W(i,t) < x\bigr] \ge x.$$

Let $v \in V$, and let $C = \{c(u) : u \in N(v)\}$ be the set of scheduling colours in the neighbourhood of v; note that $c(v) \notin C$. Let $n = |C|$ and $k = c(v)$. Summing over all possible values of $W(k,t)$, we have

$$\begin{aligned}
&\Pr\bigl[\text{node } v \text{ is active at time } t\bigr] \\
&= \Pr\bigl[w(v,t) > w(u,t) \text{ for all } u \in N(v)\bigr] \\
&= \Pr\bigl[W(k,t) > W(i,t) \text{ for all } i \in C\bigr] \\
&= \sum_{j=0}^{k-1} \Pr\Bigl[W(k,t) = \frac{j}{k}\Bigr] \cdot \Pr\Bigl[\frac{j}{k} > W(i,t) \text{ for all } i \in C\Bigr] \\
&\ge \sum_{j=0}^{k-1} \frac{1}{k}\Bigl(\frac{j}{k}\Bigr)^n = \frac{1}{k^{n+1}}\Bigl(\sum_{j=1}^{k} j^n\Bigr) - \frac{1}{k} \\
&\ge \frac{1}{k^{n+1}} \int_0^k x^n \, dx - \frac{1}{k} = \frac{1}{n+1} - \frac{1}{k}.
\end{aligned}$$

Moreover, $n \le \deg(v)$ and $k \ge R(\deg(v)) \ge (\deg(v)+1)/\varepsilon$. Therefore node v is active at time t with probability at least

$$\frac{1}{n+1} - \frac{1}{k} \ge \frac{1-\varepsilon}{\deg(v)+1} = h(v).$$

5.6 Complete Schedule

In Sect. 5.5 we defined the schedule for time interval $T(\emptyset)$. As such, this does not yet constitute a valid fractional graph colouring—indeed, it cannot be the case, as $T(\emptyset)$ is far too short.

However, we can now easily construct a valid solution by repeating the solution that we defined for $T(\emptyset)$. Define

$$H(v) = \left\lceil \frac{1}{\beta h(v)} \right\rceil. \tag{2}$$

Now the schedule $s(v)$ of node v is defined as follows: repeat the schedule defined for $T(\emptyset)$ for $H(v)$ times.

More formally, let $t > 0$. If $t \le \beta$, we have defined in Sect. 5.5 whether v is active at time t. Otherwise $t = i\beta + t'$, where $t' \in T(\emptyset)$ and $i \in \mathbb{N}$. If $i \ge H(v)$, node v is inactive. Otherwise node v is active at time t iff it is active at time t'.

Lemma 3. *Each node $v \in V$ is active for at least 1 time unit within time interval $(0, \beta H(v))$.*

Proof. Follows from Lemma 2 and (2).

Lemma 4. *If the maximum degree of G is Δ, then the length of the schedule is at most $\alpha(\Delta + 1)$.*

Proof. Let $v \in V$. We have

$$\beta H(v) \leq \frac{1}{h(v)} + \beta = \frac{\deg(v) + 1}{1 - \varepsilon} + \beta \leq \frac{\Delta + 1}{1 - \varepsilon} + \beta \leq \frac{1 + \beta}{1 - \varepsilon}(\Delta + 1) \leq \alpha(\Delta + 1).$$

That is, after time $\alpha(\Delta + 1)$, node v is no longer active.

This concludes the proof of Theorem 1—we have designed an algorithm that only needs one communication round, yet it yields a fractional graph colouring of length at most $\alpha(\Delta + 1)$.

6 Proof of Theorem 2

The theorem holds even if f assigns unique identifier from the set $\{1, 2, \ldots, n\}$, where n is the number of nodes in G_d. The proof uses the following lemma.

Lemma 5 (Bollobás [1]). *For any given integers $d \geq 3$ and $g \geq 3$, there exists a d-regular graph G with n nodes and girth at least g such that any independent set has size at most $O(n \log(d)/d)$.*

Let $\mathcal{F}$ be the family of d-regular graphs. Let $\mathcal{A}$ be a deterministic algorithm, with running time T, that finds a fractional graph colouring for any graph in $\mathcal{F}$. Now let $G = (V, E)$ be a d-regular graph with girth $g \geq 2T+1$ obtained from Lemma 5; we have $G \in \mathcal{F}$. Each independent set I of G has size at most $c|V| \log(d)/d$, for some constant c. Thus any fractional graph colouring of G has length at least $d/(c \log d)$. Choose a bijection $f \colon V \to \{1, 2, \ldots, |V|\}$.

If we run algorithm $\mathcal{A}$ on G with identifiers given by f, the output is a fractional graph colouring x of length at least $d/(c \log d)$. In particular there must be a node $v^* \in V$ that is active at time $t \geq d/(c \log d)$. Moreover, the radius-T neighbourhood of v^* is a d-regular tree, as G was a high-girth graph.

Now let $G' = (V', E')$ be the bipartite double cover of G. That is, for each node v of G we have two nodes v_1 and v_2 in G', and for each edge $\{u, v\}$ of G we have two edges $\{u_1, v_2\}$ and $\{u_2, v_1\}$ in G'. There is a covering map $\phi \colon V' \to V$ that maps $v_1 \mapsto v$ and $v_2 \mapsto v$; let $\{v_1^*, v_2^*\} = \phi^{-1}(v^*)$. Graph G' has the following properties.

1. Graph G' is bipartite; therefore there is a fractional graph colouring x' in G' with $\ell(x') = 2$.
2. Graph G' is d-regular; that is, $G' \in \mathcal{F}$.
3. The radius-T neighbourhood of $v_1^* \in V'$ is a d-regular tree.
4. The number of nodes is $|V'| = 2|V|$.

To prove the theorem, it is sufficient to show that we can choose the identifiers for $G' \in \mathcal{F}$ so that $\mathcal{A}$ outputs a fractional graph colouring of length $\Omega(d/\log d)$. To this end, observe that we can choose a bijection $f' \colon V' \to \{1, 2, \ldots, |V'|\}$ so that the radius-T neighbourhood of v_1^* in (G', f') is isomorphic to the radius-T neighbourhood of v^* in (G, f). Now apply $\mathcal{A}$ to (G', f'). By construction, the local output of v_1^* in (G', f') equals the local output of v^* in (G, f); in particular, the length of the schedule x' constructed by $\mathcal{A}'$ is $\Omega(d/\log d)$.

7 Discussion

We have shown that the fractional graph colouring problem can be solved very quickly in a distributed setting—if and only if we do not impose artificial restrictions on the size of the local outputs.

More generally, we can approach scheduling problems from the following perspective. We have three parameters:

1. T, the running time of the distributed algorithm,
2. ℓ, the length of the schedule (objective function),
3. κ, the maximum number of disjoint time intervals in the schedule of a node.

Now for the sake of concreteness, let us focus on the case of bounded-degree graphs, i.e., $\Delta = O(1)$. Our work shows that we can keep any two of T, ℓ, and κ constant, but not all three of them:

1. $T = O(1)$ and $\kappa = O(1)$: trivial, set $s(v) = (f(v), f(v) + 1]$.
2. $\kappa = O(1)$ and $\ell = O(1)$: easy, find an $O(1)$-colouring c and set $s(v) = (c(v), c(v) + 1]$.
3. $T = O(1)$ and $\ell = O(1)$: possible, using Theorem 1.
4. $T = O(1)$, $\ell = O(1)$, and $\kappa = O(1)$: impossible. Now we have an *LCL*-problem. It is easy to see that the problem cannot be solved with an order-invariant local algorithm (consider a cycle), and hence the result by Naor and Stockmeyer [8] implies that the problem cannot be solved with any local algorithm.

Acknowledgements. We thank the anonymous reviewers for their helpful comments. This work has been partially supported by the European Union under contract number ICT-2009-258885 (SPITFIRE), the Academy of Finland, Grants 132380 and 252018, the Research Funds of the University of Helsinki, and the Finnish Cultural Foundation.

References

1. Bollobás, B.: The independence ratio of regular graphs. Proceedings of the American Mathematical Society 83(2), 433–436 (1981)
2. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Information and Control 70(1), 32–53 (1986)

3. Göös, M., Hirvonen, J., Suomela, J.: Lower bounds for local approximation. In: Proc. 31st Symposium on Principles of Distributed Computing, PODC 2012 (2012)
4. Hirvonen, J., Suomela, J.: Distributed maximal matching: greedy is optimal. In: Proc. 31st Symposium on Principles of Distributed Computing, PODC 2012 (2012)
5. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proc. 23rd Symposium on Principles of Distributed Computing (PODC 2004), pp. 300–309. ACM Press, New York (2004)
6. Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: Proc. 17th Symposium on Discrete Algorithms (SODA 2006), pp. 980–989. ACM Press, New York (2006)
7. Kuhn, F., Moscibroda, T., Wattenhofer, R.: Local computation: Lower and upper bounds (2010) (manuscript) arXiv:1011.5470 [cs.DC]
8. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM Journal on Computing 24(6), 1259–1277 (1995)
9. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia (2000)
10. Shearer, J.B.: A note on the independence number of triangle-free graphs. Discrete Mathematics 46(1), 83–87 (1983)
11. Suomela, J.: Survey of local algorithms. ACM Computing Surveys (2011), http://www.cs.helsinki.fi/local-survey/ (to appear)

An Algorithm for Online Facility Leasing*

Peter Kling, Friedhelm Meyer auf der Heide, and Peter Pietrzyk

Heinz Nixdorf Institute & Computer Science Department,
University of Paderborn, 33095 Paderborn, Germany
{peter.kling,fmadh,peter.pietrzyk}@upb.de

Abstract. We consider an online facility location problem where clients arrive over time and their demands have to be served by opening facilities and assigning the clients to opened facilities. When opening a facility we must choose one of K different lease types to use. A lease type k has a certain lease length l_k. Opening a facility i using lease type k causes a cost of f_i^k and ensures that i is open for the next l_k time steps. In addition to costs for opening facilities, we have to take connection costs c_{ij} into account when assigning a client j to facility i. We develop and analyze the first online algorithm for this problem that has a time-independent competitive factor.

This variant of the online facility location problem was introduced by Nagarajan and Williamson [7] and is strongly related to both the online facility problem by Meyerson [5] and the parking permit problem by Meyerson [6]. Nagarajan and Williamson gave a 3-approximation algorithm for the offline problem and an $O(K \log n)$-competitive algorithm for the online variant. Here, n denotes the total number of clients arriving over time. We extend their result by removing the dependency on n (and thereby on the time). In general, our algorithm is $O(l_{\max} \log(l_{\max}))$-competitive. Here $l_{\max}$ denotes the maximum lease length. Moreover, we prove that it is $O(\log^2(l_{\max}))$-competitive for many "natural" cases. Such cases include, for example, situations where the number of clients arriving in each time step does not vary too much, or is non-increasing, or is polynomially bounded in $l_{\max}$.

1 Introduction

Consider a company that runs a distributed service on a network. In order to provide this service, the company has to choose a set of nodes to become service providers in such a way that they can be easily accessed by customer nodes. The nodes in the network do not belong to the company and thus have to be leased before they can be used to provide a service. There are various leases of different costs and durations. Once a lease expires, the node no longer is able to provide the service. In order to use the node again a new lease must be bought. The customer nodes can freely use any node's service as long as there is an active lease for this node. The costs of using it are proportional to the distance (latency) between customer node and service-providing

* This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and by the Graduate School on Applied Network Science (GSANS).

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 61–72, 2012.

node. This means that on the one hand the company wants to buy leases for nodes as seldom as possible, while on the other hand it wants to make sure that customer nodes are not too far away from currently leased nodes. The main problem the company is faced with is the unpredictable behavior of the customer nodes: It is not known which nodes at what point in time will request the service. Thus, the company might buy long and expensive leases for some nodes, just to realize later on that no more requests are issued to those nodes.

The problem described above is known as the *facility leasing problem*. The service providing nodes are referred to as facilities, while the customer nodes are called clients. Its offline variant (the information about the service request is known beforehand) was introduced by Anthony and Gupta [1] and is a variant of the well studied metric uncapacitated facility location problem. The online variant was presented by Nagarajan and Williamson in [7]. They introduced a constant factor approximation algorithm for the offline version and an $O(K \log n)$ factor approximation for the online version, where K is the number different leases and n is the number of clients. The facility leasing problem is strongly related to two problems introduced by Meyerson: The *online facility location problem* [5] and the *parking permit problem* [6]. In the case of the first one we have a facility leasing problem with only one lease type of infinite length, and in the second case there are multiple lease types, but no underlying facility location problem. Building upon the work of Meyerson, Fotakis presented various results for online facility location in [3]. He gave a lower bound of $O(\frac{\log n}{\log \log n})$ and showed that it is tight. This lower bound can not be directly applied to the online facility leasing problem, since in Fotakis' and Meyersons' problem variant facilities can only be opened at positions of clients that have already issued request before. This means that there are strong restrictions at what point in time a facility can be leased. There are no such restriction in the online facility leasing problem considered in this paper.

Our Contribution. We introduce the first online algorithm for the online facility leasing problem with a competitive ratio that does neither depend on the number of time steps nor on the number of clients requesting the service. The previous result by Nagarajan and Williamson [7] yields a competitive factor of $O(K \log n)$, where n is the number of clients and (assuming that at least one client arrives each time step) also an upper bound on the number of time steps. The competitive factor of Nagarajan and Williamson [7] can be expressed as $O(\log(l_{\max}) \log n)$ with $l_{\max}$ denoting the length of the longest lease type (since $\log(l_{\max})$ can be seen as an upper bound for K). Our algorithm allows us to replace the $\log n$ factor of Nagarajan and Williamson [7] with $l_{\max}$ in general and with $\log(l_{\max})$ for many "natural" special cases of client arrival sequences. These cases include, e.g., all instances where the number of clients arriving in each time step does vary only by a constant factor, or is non-increasing, or is polynomially bounded in $l_{\max}$. Thus we have an $O(l_{\max} \log(l_{\max}))$-competitive algorithm in general and, which is our main contribution, an $O(\log^2(l_{\max}))$-competitive algorithm for the arrival sequences described above. To be more precise, the approximation factor of our algorithm is $O(\log(l_{\max}) H_{l_{\max}})$, where $H_{l_{\max}} := \sum_{i=1}^{l_{\max}} (|D_i| / (\sum_{j=1}^{i} |D_j|))$ with D_i denoting the set of clients arriving in time step i. This means that our approximation guarantee depends not on the absolute numbers of clients arriving in each time step, but on the relationship (as defined by $H_{l_{\max}}$) between those numbers. While our

algorithm is not a direct improvement on Nagarajan and Williamson [7] (since we state our approximation factor using $l_{\max}$ and not K), our algorithm has the advantage that its competitive ratio is not dependent on the number of time steps.

2 Problem Definition and Notation

We consider the facility leasing problem FACILITYLEASING, a facility location variant introduced by Nagarajan and Williamson [7]. As is typical for facility location problems, we are given a set F of m facilities and a set D of n clients. Our goal is to minimize the costs of serving the clients' demands by opening facilities (which incurs costs) and assigning each client to a nearby open facility (which incurs costs that are proportional to the distance between the client and the facility). However, in contrast to the classical facility location model, there is a notion of (discrete) time. Clients do not arrive all at once. Instead, at time $t \in \mathbb{N}$ a subset $D_t \subseteq D$ of clients appears and these clients have a demand for the current time t only. Note that these subsets form a partition of D, i.e., a client arrives exactly once. Often, we will use $H_q := \sum_{i=1}^{q}(|D_i| / (\sum_{j=1}^{i} |D_j|))$ to describe a sequence of arriving clients. Another difference to the classical model is that opening a facility is not simply a binary decision. Instead, in order to open a facility one is required to determine the point in time the facility is going to be opened and one of the K different lease types that is to be used to open it. Each lease type k has length l_k and these lengths $l_1, l_2, \ldots, l_K$ are considered as a part of the input. We use $l_{\max} := \max_{1 \le k \le K}(l_k)$ to denote the maximal lease length. Consider a facility i opened at time t using lease type k. This facility is open for the l_k time steps during the interval $[t, t + l_k - 1]$ (let I_t^k denote this interval). Now, a client j arriving at time step t' can only be assigned to the facility i if i is open at j's arrival, i.e., $t' \in I_t^k$.

A solution has to assign each client to an eligible facility. Each time we open a facility $i \in F$ for the next l_k time steps using a lease type $k \in K$ a cost of f_i^k is charged. Moreover, assigning a client $j \in D$ to a facility $i \in F$ incurs a connection cost of c_{ij} (independent of the lease type used to open i). These connection costs can be assumed to correspond to the distance between facility i and client j. Clients and facilities reside in a metric space, such that the connection costs satisfy the following triangle inequality: $\forall\, i, i' \in F, j, j' \in D:\ c_{i'j} \le c_{ij} + c_{ij'} + c_{i'j'}$.

Figure 1 shows the linear programming (LP) formulation of this problem (and its dual). To ease the formulation of the LP and the analysis, we sometimes abuse the notation and write $(j,t) \in D$ for a client j appearing at time t. Similarly, we write $(i,k,t) \in F$ and refer to this triple as a single facility instead of a (potential) facility opened at time t using lease type k. The first sum in the objective function for FACILITYLEASING represents the costs incurred by opening facilities. Here, the indicator variable y_{ikt} tells us whether the facility i is opened at time step t with lease type k. The remaining part of the objective function represents the costs incurred by connecting each client to a facility, where the variable $x_{ikt',jt}$ indicates whether a client j that arrived at time step t is connected to facility i opened at time step t' with lease type k. While the first primal constraint guarantees that each client is connected to at least one facility, the second makes sure that each client is only connected to a facility that is open during the time step of the clients arrival.

$$
\begin{aligned}
\min \quad & \sum_{(i,k,t)\in F} f_i^k y_{ikt} + \sum_{(j,t)\in D} \sum_{(i,k,t')\in F : t\in I_{t'}^k} c_{ij} x_{ikt',jt} \\
\text{s.t.} \quad & \sum_{(i,k,t')\in F : t\in I_{t'}^k} x_{ikt',jt} \geq 1 && (j,t)\in D \\
& y_{ikt'} - x_{ikt',jt} \geq 0 && (i,k,t')\in F, (j,t)\in D \\
& x_{ikt',jt} \in \{0,1\} && (i,k,t')\in F, (j,t)\in D \\
& y_{ikt'} \in \{0,1\} && (i,k,t')\in F
\end{aligned}
$$

(a) ILP formulation of FACILITYLEASING.

$$
\begin{aligned}
\max \quad & \sum_{(j,t)\in D} \alpha_{jt} \\
\text{s.t.} \quad & \alpha_{jt} - \beta_{ikt',jt} \leq c_{ij} && (i,k,t')\in F, (j,t)\in D \\
& \sum_{(j,t)\in D} \beta_{ikt',jt} \leq f_i^k && (i,k,t')\in F \\
& \beta_{ikt',jt} \geq 0 && (i,k,t')\in F, (j,t)\in D \\
& \alpha_{jt} \geq 0 && (j,t)\in D
\end{aligned}
$$

(b) Dual of the ILP for FACILITYLEASING.

Fig. 1.

Given an instance I of FACILITYLEASING and a corresponding solution S, we refer to the total cost of the solution by $\text{cost}(S)$. We are mainly interested in the online version of the problem, i.e., when a client j appears at time t we have to connect it to an open facility without any knowledge of the future. Especially, we do not know whether in the following time steps more clients appear in j's proximity (which would favor a longer lease type) or not (which would encourage a shorter lease type). We measure the quality of algorithms by their *competitiveness*: Let $A(I)$ denote the solution of an algorithm A for problem instance I and $O(I)$ an optimal solution to I. Then, the competitiveness of A is defined as $\sup_I \frac{A(I)}{O(I)}$. We seek algorithms yielding a small competitive ratio.

2.1 Reduction to a Simplified Model Variant

The goal of this section is to show that we can transform a problem instance I of our online facility location problem FACILITYLEASING into an instance I' of a slightly simplified problem variant 2-FACILITYLEASING. In 2-FACILITYLEASING, all lease lengths are a power of two and a lease of type k may only start each l_k time steps. We show that any c-competitive algorithm for 2-FACILITYLEASING yields a $4c$-competitive algorithm for FACILITYLEASING (see Corollary 1). These results have already been proven in the context of the parking permit problem in [5] and we are just restating them for completeness' sake.

Rounding the Lease Length. We consider the problem variant FACILITYLEASING_1 of FACILITYLEASING that only allows lease types whose lengths are a power of two. Consider an instance I of FACILITYLEASING having K leases $k \in \{1,2,\ldots,K\}$. From I we construct an instance I' by rounding the lease lengths l_k to the next larger power of two. That is, the lease lengths l'_k for $k \in \{1,2,\ldots,K\}$ are defined as $l'_k := 2^{\lceil \log l_k \rceil}$. Other than that, I and I' are identical. Note that I' is an instance of both FACILITYLEASING and FACILITYLEASING_1.

Let O denote an optimal solution to I and O' an optimal solution to I'. First note that $\text{cost}(O') \leq \text{cost}(O)$, as any solution to I yields a solution to I' having the same cost. Indeed, whenever a facility of lease type k in the solution to I is opened, opening a facility of the same lease type k (but of lease length $l'_k \geq l_k$) yields a feasible solution to I'. Now, consider any solution S' to I'. From this, we can build a feasible solution

S to I as follows: Whenever a facility of lease type k is opened in S', we open two consecutive facilities of the same lease type in S. Since $2l_k \geq l'_k$, this yields a feasible solution to I. Moreover, we obviously have the relation $\text{cost}(S) = 2\,\text{cost}(S')$. We use these observations to prove the following lemma.

Lemma 1. *Given a c-approximation algorithm A' for the* FACILITYLEASING$_1$ *problem, we can construct a $2c$-approximation algorithm A for the* FACILITYLEASING *problem. Similarly, any c-competitive algorithm for the* FACILITYLEASING$_1$ *problem yields a $2c$-competitive algorithm for the* FACILITYLEASING *problem.*

Proof. Given a problem instance I for FACILITYLEASING we let A build a problem instance I' for FACILITYLEASING$_1$ by rounding the lease lengths to powers of two as described above and invoke A' on I'. The resulting solution S' is used to build solution S by opening two consecutive facilities for each facility opened in S' for I (see above). We have $\text{cost}(S) \leq 2\,\text{cost}(S') \leq 2c\,\text{cost}(O') \leq 2c\,\text{cost}(O)$. Note that this can be done in an online fashion, by mimicking the behavior of A' but opening two consecutive facilities whenever A' opens one. □

Restricting the Start of Leases. Let us consider the problem variant FACILITYLEASING$_2$ of FACILITYLEASING that allows to open facilities only at times t that are a multiple of their corresponding lease length. That is, a facility of type k may only open at times t with $t \equiv 0 \mod l_k$. Note that we do allow to open facilities belated, such that one may not be able to take advantage of the full lease length one has payed for.

First of all, consider a problem instance I of FACILITYLEASING. Obviously, any such I may be considered as a problem instance I' of FACILITYLEASING$_2$ without any changes. Moreover, given a solution S' to I', we can interpret it directly as a solution S to I. Thus, we have $\text{cost}(S) = \text{cost}(S')$. On the other hand, given optimal solutions O to I and O' to I', we have $\text{cost}(O') \leq 2\,\text{cost}(O)$. Indeed, any solution S to I yields a feasible solution S' to I' with $\text{cost}(S') \leq 2\,\text{cost}(S)$ as follows: Consider any facility of type $k \in \{1,2,\ldots,K\}$ opened in S at time t. For this facility, we open up to two consecutive facilities of the same lease type k in S'. The first facility is opened at time $t_1 := t - (t \mod l_k)$, the second at time $t_2 := t_1 + l_k$ if necessary. This yields a feasible solution S' to I' with cost $\text{cost}(S') \leq 2\,\text{cost}(S)$. We get the following lemma.

Lemma 2. *Given a c-approximation algorithm A' for the* FACILITYLEASING$_2$ *problem, we can construct a $2c$-approximation algorithm A for the* FACILITYLEASING *problem. Similarly, any c-competitive algorithm for the* FACILITYLEASING$_2$ *problem yields a $2c$-competitive algorithm for the* FACILITYLEASING *problem.*

Proof. Algorithm A is identical to A'. Run on a problem instance I of FACILITYLEASING, which we interpret as an identical instance I' of FACILITYLEASING$_2$, we get a solution S' to I' which can be immediately interpreted as a solution S to I (see above). We get $\text{cost}(S) = \text{cost}(S') \leq c\,\text{cost}(O') \leq 2c\,\text{cost}(O)$. □

Combining Both Variants. The model variant 2-FACILITYLEASING of our online facility location problem FACILITYLEASING combines the simplifications of both FACILITYLEASING$_1$ and FACILITYLEASING$_2$. That is, we allow only lease lengths that are a power of two and restrict the starting points of facilities of type k to multiples of l_k. By combining the results from Lemma 1 and Lemma 2, one easily gets.

Corollary 1. *Given a c-approximation algorithm A' for the* 2-FACILITYLEASING *problem, we can construct a* $4c$*-approximation algorithm A for the* FACILITYLEASING *problem. Similarly, any c-competitive algorithm for the* 2-FACILITYLEASING *problem yields a* $4c$*-competitive algorithm for the* FACILITYLEASING *problem.*

3 Algorithm Description

Let us describe our online algorithm for the 2-FACILITYLEASING problem. In the beginning, all facilities are closed. At the arrival of the client set D_t at time t, our algorithm assigns these clients to open facilities to satisfy their demands, opening new facilities if necessary. The costs charged for this step comprise the corresponding connection cost c_{ij} for assigning the clients $j \in D_t$ to open facilities $i \in F$ and the opening cost of newly opened facilities (of a certain lease type). Remember that in this simplified problem variant, we open facilities using lease type k only at times t that are a multiple of the corresponding lease length l_k. That is, only at times t with $t \equiv 0 \mod l_k$. Especially, when we say we open facility $(i,k,t) \in F$, we can only make use of it up to the the next time $t' > t$ that is a multiple of l_k. This means we may open a facility belatedly, such that we can not take advantage of the full lease length we payed for. Note that for a fixed facility i and lease type k we get a partition of the time horizon into intervals of length exactly l_k. These intervals can be identified with the corresponding facility i of lease type k which may serve clients in this interval. Now, for any time t we get exactly one interval I_i^k for each facility i and lease type k that can be used by a client appearing at time t. It remains to specify which pair (i,k) is chosen to satisfy a client's demand.

Our algorithm is based on an approximation algorithm by Jain and Vazirani [4] for the classical facility location algorithm. Their algorithm uses a primal-dual approach to compute a 3-approximation, and we make use of a similar approach in each single time step. In each time step t our algorithm operates in two phases, similar to the algorithm from Jain and Vazirani for the static facility location problem. In the first phase, the clients essentially bid towards the facilities (or more exactly, towards the intervals I_i^k). In the second phase, we use the triangle inequality to choose a cheap subset of facilities (intervals I_i^k) to actually open and assign clients to. In contrast to [4], we have to cope with the problem to build a good solution for a facility location problem starting from a partial solution (earlier arrived clients). This is similar to Nagarajan and Williamson [7], however, our subproblem is much more complex, as we consider all newly arrived clients simultaneously (instead of one after the other).

First Phase. For each client $j \in D_{\leq t} := \bigcup_{t' \leq t} D_{t'}$ that arrived at time t or before, we introduce a potential α_{jkt} that starts at zero and is continuously increased (concurrently and at the same rate for each client). Each of these potentials is reset to zero in each round. To simplify notation, let us define $(x)_+ := \max(x,0)$. For any facility i of lease type k we maintain the invariant $f_i^k \geq \sum_{j \in D_{\leq t}} (\alpha_{jkt} - c_{ij})_+$ (INV1). Whenever equality is reached for some facility i and lease type k, we temporarily open i using lease type k. As soon as $\alpha_{jkt} \geq c_{ij}$ for a client $j \in D_{\leq t}$ and a (temporarily or permanently) open facility i of lease type k, we stop increasing α_{jkt}. If $j \in D_t$ (i.e., j is a newly arrived client), we connect j to i and furthermore set $\hat{\alpha}_j := \alpha_{jtk}$ (these $\hat{\alpha}_j$ correspond to the dual variables α_{jt} in the ILP from Figure 1, the t given

implicitly by the relation $j \in D_t$). As a second invariant, we ensure that in no time step t' an $\alpha_{jkt'}$ is increased beyond $\hat{\alpha}_j$ (INV2).

Second Phase. In this phase we build K different conflict graphs, one for each lease type k. The nodes of the graph for lease type k are given by temporarily and permanently opened facilities i of lease type k. There is an edge between two nodes i and i' if and only if there is some client $j \in D_{\leq t}$ with $\alpha_{jkt} > \max(c_{ij}, c_{i'j})$. We say that the facilities i and i' are in conflict. Now, for each conflict graph we compute a maximal independent set (MIS) and open the facilities in the MIS permanently (while closing the remaining temporarily opened facilities). If for a client $j \in D_t$ (i.e., newly arrived clients) the facility i it was connected to during the first phase is not in a member of a MIS, j is reconnected to a neighbor of i that is a member of a MIS (i.e., permanently open).

4 Analysis

Here we prove that the algorithm described in the previous section is $(3+K)H_{l_{\max}}$-competitive with respect to 2-FACILITYLEASING, the simplified online facility location problem variant described in Section 2.1. Remember that $l_{\max}$ denotes the maximum lease length. Moreover, note that it is sufficient to consider the first $l_{\max}$ time steps: at time $l_{\max}$ all facilities must be closed, since for any $k \in \{1,2,\ldots,K\}$ we have $l_{\max} \equiv 0 \mod l_k$. Let us partition the time horizon into rounds $\tau_i := \{(i-1)l_{\max}, \ldots, il_{\max}-1\}$ of length $l_{\max}$. By the above observation, these rounds yield independent subproblems, each of length $l_{\max}$. We continue to show that the solution of our algorithm is $(3+K)H_{l_{\max}}$-competitive in each such round. As the costs over all rounds are additive, this yields that it is $(3+K)H_{l_{\max}}$-competitive for the complete problem.

Our analysis follows the typical idea of the analysis of a primal-dual algorithm, similar to, e.g., the analysis of Jain and Vazirani [4]. The values $\hat{\alpha}_j$ computed by our algorithm correspond to the dual variables of the ILP formulation in Figure 1. First of all, we show that the sum of all $\hat{\alpha}_j$ times $(3+K)$ is an upper bound for the cost of the solution produced by our algorithm (Lemma 3). Next, we consider the $\hat{\alpha}_j$ as a (possibly infeasible) solution to the dual program of the FACILITYLEASING ILP. We prove that by scaling this solution down by a suitable factor, we get a feasible solution to the dual program (Lemma 4). By the weak duality theorem, multiplying both factors yields the final competitive factor (Theorem 1).

Upper Bounding the Solution. The following lemma upper bounds the cost of the solution produced by our algorithm by $(3+K)\sum_{j \in D} \hat{\alpha}_j$. The basic idea is to exploit the triangle inequality to show that $3\sum_{j \in D} \hat{\alpha}_j$ is a bound on the total connection cost and that our algorithm ensures that each $\hat{\alpha}_j$ is used at most K times to cover the complete costs for opening facilities.

Lemma 3. *The cost of the primal solution produced by our algorithm can be bounded from above by* $(3+K)\sum_{j \in D} \hat{\alpha}_j$.

Proof. We bound the connection costs of the clients and the opening costs of facilities separately. The $\hat{\alpha}_j$ value of a client j is computed in step t of j's arrival during the

first phase of our algorithm. During this phase, j is either connected to a facility i that was already (permanently) opened at time $t' < t$ or one that was temporarily opened at the current time t. In both cases its $\hat{\alpha}_j$ value was set such that it can cover at least the distance c_{ij} between i and j. If i remains in one of the MIS computed in the second phase of our algorithm, this guarantees that j is assigned to a facility i' such that $\hat{\alpha}_j$ is an upper bound on the client's connection costs $c_{i'j}$. Otherwise, if i is no longer in any MIS at the end of phase two, Proposition 1 (see below) exploits the metric property of our facility location problem and yields that j is assigned to a facility i' such that $3\hat{\alpha}_j$ is an upper bound on the client's connection costs $c_{i'j}$.

Now, consider the facility costs and fix a facility i of lease type k that is permanently opened at some time t by our algorithm. As (i,k) is opened permanently at time t in the second phase, it must have been temporarily opened in the first phase. Thus, by definition of the algorithm, invariant INV1 must hold with equality, that is $f_i^k = \sum_{j \in D_{\leq t}} (\alpha_{jkt} - c_{ij})_+$. Consider these *bids* $(\alpha_{jkt} - c_{ij})_+$ of clients $j \in D_{\leq t}$ to facility i of lease type k. Note that all non-zero bids of clients j at the current time are guaranteed to be used by facility (i,k) only, as (i,k) must have been in the corresponding MIS for lease type k. Moreover, note that for a single client that arrived at time t, all its bids given to (and used by) facilities of type k sum up to at most $\hat{\alpha}_j$, as any $\alpha_{jkt'}$ with $t' > t$ stops increasing as soon as a corresponding open facility (or $\hat{\alpha}_j$) is reached. Together, this yields that the total costs for opening facilities of type k in the solution produced by our algorithm is upper bounded by $\sum_{j \in D} \hat{\alpha}_j$. As there are K different lease types, together with the bound on the connection costs we get the lemma's statement. □

The following proposition exploits the triangle inequality of our metric facility location problem and can be proven completely analogue to [4, Lemma 5].

Proposition 1. *For each client j that is reconnected in the second phase to a facility i, we have $\hat{\alpha}_j \geq \frac{1}{3} c_{ij}$.*

Proof. Let i' be the facility that client j was connected to in the first phase of the algorithm. There must be a client j' that is responsible for the conflict between facility i and i'. We have that $\alpha_{j'} \geq c_{i'j}$, $\alpha_{j'} \geq c_{ij}$ and $\alpha_j \geq c_{i'j}$. Let s_i resp. $s_{i'}$ be the points in time where i resp. i' are temporarily opened. We know that $\alpha_{j'} \leq \min(s_i, s_{i'})$ since j' was contributing to both facilities, and that $\hat{\alpha}_j \geq s_{i'}$ since j was connected to i'. Plugging this information into the triangle inequality $c_{ij} \leq c_{i'j} + c_{i'j'} + c_{ij'}$ yields the proposition. □

Scaling the Dual Variables for Feasibility. For the second part of the proof, it remains to scale down the dual solution represented by the $\hat{\alpha}_j$ such that we obtain a feasible solution. Before we do so, we need another proposition based on the triangle inequality. In spirit, it is similar to [7, Lemma 5], but has a slightly more involved proof due to the fact that we have to consider multiple $\hat{\alpha}_j$'s that increase simultaneously.

Proposition 2. *Given a client l that arrived in time step t and a facility i of type k for any client j that arrived before time t, we have $\alpha_{jkt} - c_{ij} \geq \hat{\alpha}_l - 2c_{ij} - c_{il}$.*

Proof. Showing that $\alpha_{jkt} + c_{ij} + c_{il} \geq \hat{\alpha}_l$ proves the proposition. Since for $\alpha_{jkt} \geq \hat{\alpha}_l$ the statement trivially holds, we assume the contrary. This means that client j reached an open facility i' (and thus its α_{jkt} stopped increasing) before $\hat{\alpha}_l$ was fixed (i.e., α_{lkt} stopped increasing). Since α_{lkt} stops increasing once it is large enough to cover the distance between i' and l and this distance is at most $\alpha_{jkt} + c_{ij} + c_{il}$, the proposition follows. □

Before we continue with the Lemma 4 and its proof, let us define N_t to be the number of clients that have arrived until time t (i.e., $N_t := |D_1| + |D_2| + \ldots + |D_{t-1}|$) and note that

$$N_t = N_t \frac{|D_t|}{|D_t|} = \sum_{j \in D_t} \frac{N_t}{|D_t|} = \frac{N_t}{|D_t|} \sum_{j \in D_t} 1. \tag{1}$$

For the prior defined series H_q, it holds that

$$\sum_{t=1}^{t^*} 2h_t \sum_{t'=1}^{t-1} \sum_{j \in D_{t'}} c_{ij} = \sum_{t=1}^{t^*} 2 \sum_{j \in D_t} c_{ij}(H_{t^*} - H_t), \tag{2}$$

which can be easily seen by observing that $\sum_{i=1}^{q} \sum_{j=1}^{i-1} x_j h_i = \sum_{i=1}^{q} x_i (H_q - H_i)$ holds for any series and any coefficients x_i. Given these tools, we are now ready to formulate and prove Lemma 4, which essentially shows that we get a feasible solution to the dual program of our problem if we scale the $\hat{\alpha}_j$ by a factor of $\frac{1}{H_{l_{\max}}}$. To ease notation in the following, whenever we consider a facility i of lease type k at time t^*, any time steps t we speak of are assumed to lie in the the corresponding time interval of (i,k,t^*) (all other time steps are of no interest with respect to the constraints of the dual program).

Lemma 4. *For any facility i and lease type k at time t^* we have $\sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j / 2H_{t^*} - c_{ij}) \leq f_i^k$, where $H_q := \sum_{i=1}^{q} \frac{|D_i|}{\sum_{j=1}^{i} |D_j|}$.*

Proof. Remember INV1 of our algorithm (see algorithm description in Section 3). It states that the sum of bids towards a facility at any point in time does never exceed its opening costs. Thus, for any time step t^* we have

$$\begin{aligned}
f_i^k &\geq \sum_{j \in D_{\leq t^*}} (\alpha_{jkt^*} - c_{ij})_+ = \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il})_+ + \sum_{j \in D_{<t^*}} (\alpha_{jkt^*} - c_{ij})_+ \\
&\geq \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il}) + \sum_{j \in D_{<t^*}} (\alpha_{jkt^*} - c_{ij}) = \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il}) + \sum_{t=1}^{t^*-1} \sum_{j \in D_t} (\alpha_{jkt^*} - c_{ij}) \\
&\geq \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il}) + \sum_{t=1}^{t^*-1} \sum_{j \in D_t} (\hat{\alpha}_{l^*} - c_{il^*} - 2c_{ij}) \qquad \text{(Prop. 2 and } l^* := \arg\max(\hat{\alpha}_l - c_{il})\text{)} \\
&= \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il}) + \sum_{t=1}^{t^*-1} \sum_{j \in D_t} (\hat{\alpha}_{l^*} - c_{il^*}) - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij} \\
&= \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il}) + (\hat{\alpha}_{l^*} - c_{il^*}) \frac{N_{t^*}}{|D_{t^*}|} \sum_{j \in D_{t^*}} 1 - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij} \\
&= \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il}) + \frac{N_{t^*}}{|D_{t^*}|} \sum_{j \in D_{t^*}} (\hat{\alpha}_{l^*} - c_{il^*}) - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij}
\end{aligned}$$

$$\geq \sum_{l \in D_{t^*}} (\hat{\alpha}_l - c_{il}) + \frac{N_{t^*}}{|D_{t^*}|} \sum_{j \in D_{t^*}} (\hat{\alpha}_j - c_{ij}) - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij}$$

$$= \left(1 + \frac{N_{t^*}}{|D_{t^*}|}\right) \sum_{j \in D_{t^*}} (\hat{\alpha}_j - c_{ij}) - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij} = \left(\frac{N_{t^*} + |D_{t^*}|}{|D_{t^*}|}\right) \sum_{j \in D_{t^*}} (\hat{\alpha}_j - c_{ij}) - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij}.$$

The above inequality holds for each $t \in \{1, \ldots, t^*\}$. Dividing each such inequality by $\frac{N_t + |D_t|}{|D_t|}$ yields the following set of inequalities:

$$\frac{|D_{t^*}|}{N_{t^*} + |D_{t^*}|} f_i^k \geq \sum_{j \in D_{t^*}} (\hat{\alpha}_j - c_{ij}) - 2 \frac{|D_{t^*}|}{N_{t^*} + |D_{t^*}|} \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij}$$

$$\frac{|D_{t^*-1}|}{N_{t^*-1} + |D_{t^*-1}|} f_i^k \geq \sum_{j \in D_{t^*-1}} (\hat{\alpha}_j - c_{ij}) - 2 \frac{|D_{t^*-1}|}{N_{t^*-1} + |D_{t^*-1}|} \sum_{t=1}^{t^*-2} \sum_{j \in D_t} c_{ij}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$\frac{|D_2|}{N_2 + |D_2|} f_i^k \geq \sum_{j \in D_2} (\hat{\alpha}_j - c_{ij}) - 2 \frac{|D_2|}{N_2 + |D_2|} \sum_{t=1}^{1} \sum_{j \in D_t} c_{ij}$$

$$\frac{|D_1|}{N_1 + |D_1|} f_i^k \geq \sum_{j \in D_1} (\hat{\alpha}_j - c_{ij}) - 0.$$

Adding up these t^* inequalities yields

$$\left(\frac{|D_1|}{N_1 + |D_1|} + \ldots + \frac{|D_{t^*}|}{N_{t^*} + |D_{t^*}|}\right) f_i^k \geq \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - c_{ij}) - \sum_{t=1}^{t^*} 2 \frac{|D_t|}{N_t + |D_t|} \sum_{t'=1}^{t-1} \sum_{j \in D_{t'}} c_{ij}.$$

Due to Inequality (2) we have

$$H_{t^*} f_i^k \geq \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - c_{ij}) - \sum_{t=1}^{t^*} 2 \sum_{j \in D_t} c_{ij} (H_{t^*} - H_t)$$

$$= \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - 2 H_{t^*} c_{ij}) + \sum_{t=1}^{t^*} \sum_{j \in D_t} 2 c_{ij} \left(H_t - \frac{1}{2}\right)$$

$$\geq \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - 2 H_{t^*} c_{ij})$$

Dividing by $2H_{t^*}$ yields $\sum_{t=1}^{t^*} \sum_{j \in D_t} \left(\frac{\hat{\alpha}_j}{2H_{t^*}} - c_{ij}\right) \leq \frac{f_i^k}{2} \leq f_i^k$. □

Finally, by combining our results from Corollary 1, Lemma 3 and Lemma 4, and using that our time horizon is at most $l_{\max}$ (i.e., $t^* \leq l_{\max}$), the weak duality theorem implies a competitive factor depending on the series H_k.

Theorem 1. *Our algorithm is at most* $4(3+K)H_{l_{max}}$*-competitive for the* FACILITYLEASING *problem. Here, the series* H_q *is defined by* $H_q := \sum_{i=1}^{q} \frac{|D_i|}{\sum_{j=1}^{i} |D_j|}$ *and describes the relationship between the number of clients that arrive in each step.*

The following corollaries bring the competitive factor guaranteed by Theorem 1 into a more concrete and compact form.

Corollary 2. *Our algorithm is at most* $4(3+K)l_{max} = O(\log(l_{max})l_{max})$*-competitive for the* FACILITYLEASING *problem.*

Corollary 3. *If for each round, the number of clients at any time t does vary by at most a constant factor, is non-increasing, or bounded from above by a polynomial in* l_{max}*, the competitive factor of our algorithm becomes at most* $O(K\log(l_{max})) = O(\log^2(l_{max}))$*-competitive for the* FACILITYLEASING *problem.*

While Corollary 3 arguably covers the most interesting and realistic cases, it seems probable that one can in fact construct an instance where the bound given in Corollary 2 is tight. Based on the convergence behavior of the series H_k, instances where the number of arriving clients increases at least exponentially seem the most difficult and challenging for an online algorithm.

5 Conclusion and Future Work

We gave the first algorithm for the online facility leasing problem FACILITYLEASING that has a time-independent competitive factor of $O(l_{\max}\log(l_{\max}))$ in general and $O(\log(l_{\max})^2)$ in many common cases. The competitive factor can be upper bounded by $O(H_{l_{\max}}\log(l_{\max}))$, where $H_{l_{\max}}$ is defined by the series $H_t := \sum_{i=1}^{t}\frac{D_i}{\sum_{j}^{i} D_j}$. The $D_i \in \mathbb{N}$ represent the number of clients arriving at time step i. For an exponential increase in the number of arriving clients, e.g., $D_i = 2^i$, we have $H_t = \Theta(t)$. We conjecture that, based on this observation, it is possible to build an instance that shows that our upper bound is tight for our algorithm. Instances featuring such an exponential increase in arriving clients seem to have a unique hardness for an online algorithm: At any time t the number of arriving clients essentially matches the total number of clients that arrived up to now. Thus, in each single time step we have to solve a problem being as hard as the complete problem up to the current time. It remains an interesting problem, whether such instances are inherently difficult to handle for online algorithms, or whether this conjectured lower bound is merely limited to our online algorithm.

Our competitive bounds can also be written as $O(Kl_{\max})$ and $O(K\log(l_{\max}))$, respectively. Meyerson [6] showed a lower bound of $\Omega(K)$ on the competitive ratio of deterministic online algorithms for the parking permit problem, a special case of our online facility leasing problem FACILITYLEASING. For randomized algorithms, a lower bound of $\Omega(\log K)$ is proven. As these bounds immediately carry over to our model, one may hope to improve our bounds to $O(l_{\max}\log(K))$ and $O(\log(K)\log(l_{\max}))$ using randomization.

Another direction for possible future research includes a distributed implementation of our algorithm, similar in spirit to [2, 8]. Such distributed and local implementations, where a solution is computed not by a central authority but a network of distributed sensor nodes (e.g., in our case, the facilities and clients), have attracted much interest in recent years, and the primal-dual approach our algorithm uses has proven to be compatible with such a distributed model in other scenarios.

References

1. Anthony, B.M., Gupta, A.: Infrastructure Leasing Problems. In: Fischetti, M., Williamson, D.P. (eds.) IPCO 2007. LNCS, vol. 4513, pp. 424–438. Springer, Heidelberg (2007), doi:10.1007/978-3-540-72792-7_32, ISBN 978-3-540-72791-0
2. Blelloch, G.E., Tangwongsan, K.: Parallel approximation algorithms for facility-location problems. In: Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 315–324 (2010) ISBN 978-1-4503-0079-7
3. Fotakis, D.: A primal-dual algorithm for online non-uniform facility location. Journal of Discrete Algorithms 5(1), 141–148 (2007), doi:10.1016/j.jda.2006.03.001, ISSN 1570-8667
4. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. Journal of the ACM 48(2), 274–296 (2001) ISSN 0004-5411
5. Meyerson, A.: Online facility location. In: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 426–431. IEEE Computer Society, Washington, DC (2001) ISBN 0-7695-1390-5
6. Meyerson, A.: The parking permit problem. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 274–284. IEEE Computer Society, Washington, DC (2005), doi:10.1109/SFCS.2005.72, ISBN 0-7695-2468-0
7. Nagarajan, C., Williamson, D.P.: Offline and Online Facility Leasing. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 303–315. Springer, Heidelberg (2008), doi:10.1007/978-3-540-68891-4_21, ISBN 978-3-540-68886-0
8. Pandit, S., Pemmaraju, S.V.: Rapid randomized pruning for fast greedy distributed algorithms. In: Proceeding of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), pp. 325–334. ACM, New York (2010), doi:10.1145/1835698.1835777, ISBN 978-1-60558-888-9

Agreement in Directed Dynamic Networks*

Martin Biely[1], Peter Robinson[2], and Ulrich Schmid[3]

[1] EPFL, Switzerland
[2] Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371
[3] Technische Universität Wien, Embedded Computing Systems Group (E182/2), Vienna, Austria

Abstract. We study the fundamental problem of achieving consensus in a synchronous dynamic network, where an omniscient adversary controls the unidirectional communication links. Its behavior is modeled as a sequence of *directed* graphs representing the active (i.e. timely) communication links per round. We prove that consensus is impossible under some natural weak connectivity assumptions, and introduce vertex-stable root components as a—practical and not overly strong—means for circumventing this impossibility. Essentially, we assume that there is a short period of time during which an arbitrary part of the network remains strongly connected, while its interconnect topology keeps changing continuously. We present a consensus algorithm that works under this assumption, and prove its correctness. Our algorithm maintains a local estimate of the communication graphs, and applies techniques for detecting stable network properties and univalent system configurations. Our possibility results are complemented by several impossibility results and lower bounds, which reveal that our algorithm is asymptotically optimal.

1 Introduction

Dynamic networks, instantiated, e.g., by (wired) peer-to-peer (P2P) networks, (wireless) sensor networks, mobile ad-hoc networks and vehicular area networks, are becoming ubiquitous nowadays. The primary properties of such networks are (i) sets of participants (called processes in the sequel) that are a priori unknown and maybe time-varying, and (ii) the absence of central control. Such assumptions make it very difficult to setup and maintain the basic system, and create particular challenges for the design of robust distributed services for applications running on such dynamic networks.

A natural approach to build robust services despite mobility, churn, failures, etc. of processes is to use distributed consensus to agree system-wide on (fundamental) parameters like schedules, frequencies, etc. Although system-wide

* This work has been supported by the Austrian Science Foundation (FWF) project P20529 and S11405. Peter Robinson has also been supported in part by Nanyang Technological University grant M58110000 and Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 grant MOE2010-T2-2-082.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 73–84, 2012.

agreement indeed provides a very convenient abstraction for building robust services, it inevitably rests on the ability to efficiently implement consensus in a dynamic network.

Doing this in *wireless* dynamic networks is particularly challenging, for several reasons: First, whereas wireline networks are usually adequately modeled by means of bidirectional links, this is not the case for wireless networks: Fading phenomenons and interference [1] are local effects that affect only the receiver of a wireless link. Since the receiver of the reverse link is located in a different place in the network, it is very likely that it faces very different levels of fading and interference. Thus, wireless links are more adequately modeled by means of pairs of unidirectional links, which are considered independent of each other.

Second, wireless networks are inherently broadcast. When a process transmits, then every other process within its transmission range will observe this transmission — either by legitimately receiving the message or as some form of interference. This creates quite irregular communication behavior, such as capture effects and near-far problems [2], where certain (nearby) transmitters may "lock" some receiver and thus prohibit the reception of messages from other senders. As a consequence, wireless links that work correctly at a given time may have a very irregular spatial distribution, and may also vary heavily with time.

Finally, taking also into account mobility of processes and/or peculiarities in the system design (for example, duty-cycling is often used to conserve energy in wireless sensor networks), it is obvious that static assumptions on the communication topology, as underlying classic models like unit disc graphs, are not adequate for wireless dynamic networks.

We hence argue that such dynamic systems can be modeled adequately only by means of dynamically changing *directed* communication graphs. Since synchronized clocks are required already for basic communication in wireless systems, e.g., for transmission scheduling and sender/receiver synchronization, we also assume that the system is synchronous.

Contributions. Similar to Kuhn et al. [3], we consider consensus in a system modeled by means of a sequence of communication graphs, one for each round. In sharp contrast to existing work, our communication graphs are directed, and our rather weak connectivity assumptions do not guarantee bidirectional (multi-hop) communication between all processes.

(1) We prove that communication graphs that are weakly connected in every round are not sufficient for solving consensus, and introduce a fairly weak additional assumption that allows to overcome this impossibility. It requires that, in every round, there is exactly one arbitrary strongly connected component (called a *root component*) that has only out-going links to (some of) the remaining processes and can reach every process in the system via several hops. Since this assumption is still too weak for solving consensus, we add the requirement that, eventually, there will be a short interval of time where the processes in the root component remain the same, although the connection topology may change. We coined the term *vertex-stable root component* (for some window of limited stability) for this requirement.

(2) We provide a consensus algorithm that works in this model, and prove its correctness. Our algorithm requires a window of stability that has a size of $4D$, where D is the number of rounds required to reach all processes in the network from any process in the vertex-stable root component.

(3) We show that any consensus algorithm has to know an a priori bound on D. Since the system size n is a trivial bound on D, this implies that there is no uniform algorithm, i.e., no algorithm unaware of the size of the network, that solves consensus in our model. In addition, we establish a lower bound of D for the window of stability.

(4) We prove that neither reliable broadcast, atomic broadcast, nor causal-order broadcast can be implemented in our model without additional assumptions. Moreover, there is no algorithm that solves counting, k-verification, k-token dissemination, all-to-all token dissemination, and k-committee election.

Lacking space did not allow us to include the detailed algorithms and proofs in this paper; consult the full paper [4] for all the details.

Related Work

We are are not aware of any previous work on consensus in *directed* and *dynamic* networks with such weak connectivity requirements. This is also true for an earlier paper [5], where we assumed the existence of an underlying *static* skeleton graph (a non-empty common intersection of all communication graphs of all rounds), which had to include a *static* root component. By contrast, in this paper, we allow the graphs to be totally dynamic, except for a (sufficiently large) time window where the members (but not the topology!) of the root component are the same.

Dynamic networks have been studied intensively in distributed computing. Early work on this topic includes [6,7]. We will in the following focus on two lines of research that are closest to ours: work that models directly the underlying (evolving) communication graph, and approaches taken in the context of consensus.

There is a rich body of literature on dynamic graph models going back to [8], which also mentions for the first time modeling a dynamic graph as a sequence of static graphs, as we do. A survey on dynamic networks can be found in [9]. Recently, Casteigts et al. [10] have introduced a classification of time varying graphs, that is, a classification of the assumptions about the temporal properties of these graphs. In the full paper [4], we show that our assumption falls between two of the weakest classes considered, as we can only guarantee one-directional reachability.[1] We are not aware of any other papers considering such weak assumptions in the context of agreement.

Closest to our own work is that of Kuhn et al. [3], who also consider agreement problems in dynamic networks based on the model of [11]. This model

[1] Here, reachability does not refer to the graph-theoretic concept of reachability, but rather to the ability to eventually communicate information to another process.

is based on distributed computations organized in lock-step rounds, and states assumptions on the connectivity in each round as a separate (round) communication graph. While the focus of [11] is the complexity of aggregation problems in dynamics networks, [3] focuses on agreement problems; more specifically on the Δ-coordinated consensus problem, which extends consensus by requiring all processes to decide within Δ rounds of the first decision. In both papers, only (i) undirected graphs that are (ii) *connected in every round* are considered. In terms of the classes of [10], they are in one of the strongest classes (Class 10), which means (among other things) that each process is always reachable by every other process. Since node failures are not considered, solving consensus is always possible in this model without additional assumptions; the focus of [3] is on Δ-coordinated and simultaneous consensus and its time complexity.

Instead of considering a dynamic graph that defines which processes communicate in each round, an alternative approach is based on the (dual) idea of assuming a fully connected network of (potential) communication, and considering that communication in a round can fail. The notion of transmission failures was introduced by Santoro and Widmayer [12], who assumed dynamic transmission failures and showed that $n-1$ dynamic transmission failures in the benign case (or $n/2$ in case of arbitrary transmission failures) render any nontrivial agreement impossible. As it assumes unrestricted transmission failures (the (only) case considered in their proof are failures that affect all the transmissions of a *single* process), it does not apply to any model which considers perpetual mutual reachability of processes (e.g., [3]).

The HO-model [13] is also based on transmission failures. It relies on the collection of sets of processes a process *hears of* (i.e., receives a message from) in a round. Different system assumptions are modeled by predicates over this collection of sets. The HO-model is totally oblivious to the actual reason *why* some process does not hear from another one: Whether the sender committed a send omission or crashed, the message was lost during transmission or is simply late, or the receiver committed a receive omission. A version of the model also allowing communication to be corrupted is presented in [14]. Indeed, the HO-model is very close to our graph model, as an edge from p to q in the graph of round r corresponds to p being in the round r HO set of q.

The approach taken by Gafni [15] has some similarities to the HO-model (of which it is a predecessor), but is more focused on process failures than the work by Santoro and Widmayer. Here an oracle (a round-by-round failure detector) is considered to tell processes the set of processes they will be not be able to receive data from in the current round. Unlike the approaches discussed above, it explicitly states how rounds are implemented; nevertheless, the oracle abstracts away the actually reason for not receiving a message. So, like in the HO-model, the same device is used to describe failures and (a)synchrony.

Another related model is the perception based failure model [16,17], which uses a sequence of perception matrices (corresponding to HO sets) to express failures of processes and links. As for communication failures, the impossibility result of Santoro and Widmayer is circumvented by putting separate restrictions

on the number of outgoing and incoming links that can be affected by transmission failures [16]. Since transmission failures are counted on a per process/per round basis, agreement was shown to be possible in the presence of $O(n^2)$ total transmission failures per round.

2 Model and Preliminaries

We consider synchronous computations of a dynamic network of a fixed set of distributed processes Π with $|\Pi| = n \geqslant 2$. Processes can communicate with their current neighbors in the network by sending messages taken from some finite message alphabet $\mathcal{M}$.

Similar to the $\mathcal{LOCAL}$ model [18], we assume that processes organize their computation as an infinite sequence of lock-step rounds. For every $p \in \Pi$ and each round $r > 0$, let $S_p^r \in \mathcal{S}_p$ be the state of p at the beginning of round r; the initial state is denoted by $S_p^1 \in \mathcal{S}_p^1 \subset \mathcal{S}_p$. The round r computation of process p is determined by the following two functions that make up p's algorithm: The message sending function $M_p : \mathcal{S}_p \to \mathcal{M}$ determines the message m_p^r broadcast by p in round r, based on p's state S_p^r at the beginning of round r. We assume that some (possibly empty) message is broadcast in every round, to all (current!) neighbors of p. The transition function $T_p : \mathcal{S}_p \times 2^{(\Pi \times \mathcal{M})} \to \mathcal{S}_p$ takes p's state S_p^r at the beginning of round r and a set of pairs of process ids and messages μ_p^r. This set represents the round r messages[2] received by p from other processes in the system, and computes the successor state S_p^{r+1}. We assume that, for each process q, there is at most one $(q, m_q^r) \in \mu_p^r$ such that m_q^r is the message q sent in round r. Note that neither M_p nor T_p need to involve n, i.e., the algorithms executed by the processes may be uniform w.r.t. the network size n: Which processes a process actually receives from in round r depends solely on the underlying communication graph of round r.

To formally introduce the consensus problem, we assume some ordered set V and consider the set of possible initial states $\mathcal{S}_p^1$ (of process p) to be partitioned into $|V|$ subsets $\mathcal{S}_p^1[v]$, with $v \in V$. When p starts in a state in $\mathcal{S}_p^1[v]$, we say that v is p's input value, denoted $v_p = v$. Moreover, we assume that, for each $v \in V$, there is a (sub-)set $\mathcal{D}_p[v] \subset \mathcal{S}_p$ of decided states that is closed under p's transition function, i.e., where T_p maps any state in this subset to this subset. We say that p has decided on v when it is in some state in $\mathcal{D}_p[v]$. When p performs a transition from a state outside of the set of decided states to the set of decided states, we say that p decides. We say that an *algorithm* $\mathcal{A}$ *solves consensus* if the following properties hold in every run of $\mathcal{A}$:

Agreement: If process p decides on x_p and q decides on x_q, then $x_p = x_q$.
Validity: If a process decides on v, then v was proposed by some q, i.e., $v_q = v$.
Termination: Every process must eventually decide.

[2] We only consider messages sent in round r here, so we assume communication-closed [19] rounds.

At a first glance, solving consensus might appear easier in our model than in the classic crash failure model, where processes simply stop executing the algorithm. This is not the case, however. As in [13], we model crash failures as follows: A process q that crashes in round r is equivalent to taking away all outgoing edges of q from round $r+1$ on. While q itself can still receive messages and perform computations, the remaining processes are not influenced by q from round r on.

Communication Model. The evolving nature of the network topology is modeled as an infinite sequence of simple directed graphs $\mathcal{G}^1, \mathcal{G}^2, \ldots$, which is fixed by an adversary having access to the processes' states. For each round r, we denote the *round r communication graph* by $\mathcal{G}^r = \langle V, E^r \rangle$, where each node of the set V is associated with one process from the set of processes Π, and where E^r is the set of directed edges for round r, such that there is an edge from p to q, denoted as $(p \to q)$, *iff* q receives p's round r message (in round r). For any (sub)graph G, we will use the notation $V(G)$ and $E(G)$ to refer to the set of vertices respectively edges of G, i.e., it always holds that $G = \langle V(G), E(G) \rangle$.

To simplify the presentation, we will denote a process and the associated node in the communication graph by the same symbols and omit the set from which it is taken if there is no ambiguity. We will henceforth write $p \in \mathcal{G}^r$ and $(p \to q) \in \mathcal{G}^r$ instead of $p \in V$ resp. $(p \to q) \in E^r$.

The *neighborhood of p in round r* is the set of processes $\mathcal{N}_p^r$ that p receives messages from in round r, formally, $\mathcal{N}_p^r = \{q \mid (q \to p) \in \mathcal{G}^r\}$.

Similarly to the classic notion of "happened-before" [20], we say that a process *p (causally) influences process q in round r*, expressed by $(p \overset{r}{\rightsquigarrow} q)$ or just $(p \rightsquigarrow q)$ if r is clear from the context, iff either (i) $p \in \mathcal{N}_q^r$, i.e., if q has an incoming edge $(p \to q)$ from p in $\mathcal{G}^r$, or (ii) if $q = p$, i.e., we assume that p always influences itself in a round. We say that there is a *(causal) chain of length $k \geqslant 1$ starting from p in round r to q*, graphically denoted by $(p \overset{r[k]}{\rightsquigarrow} q)$, if there exists a sequence of not necessarily distinct processes $p = p_0, \ldots, p_k = q$ such that p_i influences p_{i+1} in round $r+i$, for all $0 \leqslant i < k$.

The *causal distance* $\mathtt{cd}^r(p,q)$ at round r from process p to process q is the length of the shortest causal chain starting in p in round r and ending in q, formally, $\mathtt{cd}^r(p,q) := \min\{k \mid (p \overset{r[k]}{\rightsquigarrow} q)\}$. Note that we assume $\mathtt{cd}^r(p,p) = 1$. The following Lemma 1 shows that the causal distance in successive rounds cannot arbitrarily decrease.

Lemma 1 (Causal distance in successive rounds). *For every round $r \geqslant 1$ and every two processes $p, q \in \Pi$, it holds that $\mathtt{cd}^{r+1}(p,q) \geqslant \mathtt{cd}^r(p,q) - 1$. As a consequence, if $\mathtt{cd}^r(p,q) = \infty$, then also $\mathtt{cd}^{r+1}(p,q) = \infty$.*

Note that, in contrast to the similar notion of dynamic distance defined in [9], the causal distance in *directed* graphs is not necessarily symmetric. Moreover, if the adversary chooses the graphs $\mathcal{G}^r$ such that not all nodes are strongly connected, the causal distance can even be infinite. In fact, even if $\mathcal{G}^r$ is strongly connected for round r (but not for rounds $r' > r$), $\mathtt{cd}^r(p,q)$ can be infinite. As we will not consider the whole communication graph to be strongly connected in this paper,

we make use of the notation of *strongly connected components (SCC)*. We write $\mathcal{C}_p^r$ to denote the (unique) SCC of $\mathcal{G}^r$ that contains process p in round r or simply $\mathcal{C}^r$ if p is irrelevant.

It is apparent that $\mathtt{cd}^r(p,q)$ and $\mathtt{cd}^r(q,p)$ may be infinite even if $q \in \mathcal{C}_p^r$. In order to be able to argue (meaningfully) about the maximal length of causal chains within an SCC, we also need some "continuity property" over rounds. This leads us to the crucial concept of a *I-vertex-stable strongly connected component*, denoted as $\mathcal{C}^I$: It requires that the set of vertices of a strongly connected component $\mathcal{C}$ remains stable throughout all rounds in the nonempty interval I. Its topology may undergo changes, but must form an SCC in every round. Formally, $\mathcal{C}^I$ being vertex-stable during I requires that $\forall p \in \mathcal{C}^I, \forall r \in I : V(\mathcal{C}_p^r) = V(\mathcal{C}^I)$. The important property of $\mathcal{C}^I$ is that information is guaranteed to spread to all vertices of $\mathcal{C}^I$ if the interval I is large enough (cf. Lemma 3).

Let the *round r causal diameter* $D^r(\mathcal{C}^I)$ of a vertex-stable SCC $\mathcal{C}^I$ be the largest causal distance $\mathtt{cd}^r(p,q)$ for any $p, q \in \mathcal{C}^I$. The *causal diameter* $D(\mathcal{C}^I)$ of a vertex-stable SCC $\mathcal{C}^I$ in I is the largest causal distance $\mathtt{cd}^x(p,q)$ starting at any round $x \in I$ that "ends" in I, i.e., $x + \mathtt{cd}^x(p,q) - 1 \in I$. If there is no such causal distance (because I is too short), $D(\mathcal{C}^I)$ is assumed to be infinite. Formally, for $I = [r, s]$ with $s \geqslant r$,[3]

$$D(\mathcal{C}^I) = \min\left\{\max\{D^x(\mathcal{C}^I) \mid x \in [r,s] \text{ and } x + D^x(\mathcal{C}^I) - 1 \leqslant s\}, \infty\right\}.$$

If $\mathcal{C}^I$ consist only of one process, then we obviously have $D(\mathcal{C}^I) = 1$. The following Lemma 2 establishes a bound for $D(\mathcal{C}^I)$ also for the general case.

Lemma 2. *Let a vertex-stable SCC $\mathcal{C}^I$ for some $I = [r, s]$ be given and let $|\mathcal{C}^I| \geqslant 2$ be the number of processes in $\mathcal{C}^I$. If $s \geqslant r + |\mathcal{C}^I| - 2$, then $D(\mathcal{C}^I) \leqslant |\mathcal{C}^I| - 1$.*

Given this result, it is tempting to assume that, for any vertex-stable SCC $\mathcal{C}^I$ with finite causal diameter $D(\mathcal{C}^I)$, any information propagation that starts at least $D(\mathcal{C}^I) - 1$ rounds before the final round of I will reach all processes in $\mathcal{C}^I$ within I. This is not generally true, however, as the following example for $I = [1, 3]$ and a vertex-stable SCC of four processes shows: If $\mathcal{G}^1$ is the complete graph whereas $\mathcal{G}^2 = \mathcal{G}^3$ is a ring, $D(\mathcal{C}^I) = 1$, but information propagation starting at round 2 does not finish by the end of round 3. However, the following Lemma 3 gives a bound on the earliest starting round that guarantees this property.

Lemma 3 (Information propagation). *Suppose that $\mathcal{C}^I$ is an I-vertex-stable strongly connected component of size $\geqslant 2$ that has $D(\mathcal{C}^I) < \infty$, for $I = [r, s]$, and let x be the maximal round where $x + D^x(\mathcal{C}^I) - 1 \leqslant s$. Then,*

(i) for every $x' \in [r, x]$, it holds that $x' + D^{x'}(\mathcal{C}^I) - 1 \leqslant s$ and $D^{x'}(\mathcal{C}^I) \leqslant D(\mathcal{C}^I)$ as well, and

(ii) $x \geqslant \max\{s - |\mathcal{C}^I| + 2, r\}$.

[3] Since I ranges from the beginning of r to the end of s, we define $|I| = s - r + 1$.

Since we will frequently require a vertex-stable SCC $\mathcal{C}^I$ that guarantees bounded information propagation also for late starting rounds, we introduce the following Definition 1.

Definition 1. *An I-vertex-stable SCC $\mathcal{C}^I$ with $I = [r, s]$ is* D-bounded *if $D \geqslant D^I(\mathcal{C}^I)$ and $D^{s-D+1}(C^I) \leqslant D$.*

3 Required Connectivity Properties

Up to now, we did not provide any guarantees on the connectivity of the network, the lack of which makes consensus trivially impossible. In this section, we will add some weak constraints on the adversary that circumvent this impossibility. Obviously, we want to avoid requesting strong properties of the network topology (such as stating that $\mathcal{G}^r$ is strongly connected in every round r), as this would reduce the applicability of our results in real networks.

As a first attempt, we could assume that, in every round r, the communication graph $\mathcal{G}^r$ is weakly connected. This, however, turns out to be too weak. Even if the adversary choses a *static* topology, it is easy to see that consensus remains impossible: Consider for example the graph that is partitioned into 3 strongly connected components $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$ such that there are only outgoing edges from $\mathcal{C}_0$ respectively $\mathcal{C}_1$ pointing to $\mathcal{C}_2$, whereas $\mathcal{C}_2$ has no outgoing edges. If all processes in $\mathcal{C}_0$ start with 0 and all processes in $\mathcal{C}_1$ start with 1, this yields a contradiction to agreement: For $i \in \{0, 1\}$, processes in $\mathcal{C}_i$ can never learn the value $1 - i$, thus, by an easy indistinguishability argument, it follows that processes in $\mathcal{C}_0$ and $\mathcal{C}_1$ must decide on conflicting values.

In order to define constraints that rule out the existence of $\mathcal{C}_0$ and $\mathcal{C}_1$ as above, the concept of *root components* proves useful: Let $\mathcal{R}^r \subseteq \mathcal{G}^r$ be an SCC that has no incoming edges from any $q \in \mathcal{G}^r \setminus \mathcal{R}^r$. We say that $\mathcal{R}^r$ is a *root component* in round r, formally: $\forall p \in \mathcal{R}^r \ \forall q \in \mathcal{G}^r \colon (q \ \rightarrow \ p) \in \mathcal{G}^r \Rightarrow q \in \mathcal{R}^r$.

Observation 1 (On root components). *Any $\mathcal{G}^r$ contains at least one and at most n root components (isolated processes), which are all disjoint. In case of a single root component $\mathcal{R}$, $\mathcal{G}^r$ is weakly connected.*

Returning to the consensus impossibility example for weakly connected graphs above, it is apparent that the two components $\mathcal{C}_0$ and $\mathcal{C}_1$ are indeed both root components. Since consensus is not solvable in this case, we assume in the sequel that there is at most *a single* root component in $\mathcal{G}^r$, for any round r. We know already [5] that this assumption makes consensus solvable if the topology (and hence the root component) is static. Since we are interested in *dynamic* networks, however, we assume in this paper that the root component may change throughout the run, i.e., the (single) root component $\mathcal{R}^r$ of $\mathcal{G}^r$ might consist of a different set of processes in every round round r. It is less straightforward to reason about the solvability of consensus in this case. However, as we will establish in Sect. 5, consensus is again impossible to solve without further constraints.

As root components are special cases of strongly connected components, we define an *I-vertex-stable root component* $\mathcal{R}^I$ as an I-vertex-stable strongly connected component that is a root component in every round $r \in I$. Clearly, all the definitions and results for vertex-stable components carry over to vertex-stable root components.

Restricting our attention to the case where exactly one vertex-stable root component $\mathcal{R}^I$ exists, it immediately follows from Observation 1 that information of any process in $\mathcal{R}^I$ propagates to all nodes in Π if I is large enough. More specifically, we can extend our notions of causal diameter of a vertex-stable SCC to the whole network: The *round r network causal diameter* D^r is the largest $\mathtt{cd}^r(p,q)$ for any $p \in \mathcal{R}^r$ and any $q \in \Pi$. Similarly to the causal diameter of a vertex-stable component of an interval, we define the *network causal diameter* D^I for an interval I as the largest round x, $x \in I$, network causal diameter that also (temporally) falls within I, i.e., satisfies $x + D^x - 1 \in I$ and hence $x + \mathtt{cd}^x(p,q) - 1 \in I$ for any $p \in \mathcal{R}^r$ and any $q \in \Pi$. It is straightforward to establish versions of Lemma 2 and 3 for root components and their causal influence.

Note that a plain I-vertex-stable root component with $I \geqslant n-1$ is always D-bounded for $D = n-1$. Our definition also allows some smaller choice of D, however.

We will show in Sect. 5 that the following Assumption 1 is indeed very weak, in the sense that many problems considered in distributed computing remain unsolvable.

Assumption 1. *For any round r, there is exactly one root component $\mathcal{R}^r$ in $\mathcal{G}^r$, and all vertex-stable root components $\mathcal{R}^I$ with $|I| \geqslant D$ are D-bounded. Moreover, there exists an interval of rounds $I = [r_{ST}, r_{ST}+d]$, with $d > 4D$, such that there is a D-bounded I-vertex-stable root component.*

4 Solving Consensus by Network Approximation

Initially, every process p has no knowledge of the network — it only knows its own input value. Any algorithm that correctly solves consensus must guarantee that, when p makes its decision, it either knows that its value has been/will be adopted by all other processes or it has agreed to take over some other process' decision value. As we have seen, p's information is only guaranteed to propagate throughout the network if p is in a I-vertex stable root component with finite network causal diameter D^I. Thus, for p to locally acquire knowledge about information propagation, it has to acquire knowledge about the (dynamic) communication graph.

We allow p to achieve this by gathering as much local information on $\mathcal{G}^s$ as possible, for every past round s. Every process p keeps track of its current graph approximation in variable A_p, which initially consists of process p, without any edges, and is broadcast and updated in every round. Ultimately, every process p will use A_p to determine whether it has been inside a vertex-stable root component for sufficiently many rounds. Given A_p, we will denote the information

contained in A_p about round s by $A_p|s$. More specifically, $A_p|s$ is the graph induced by the set of edges $E_p|s = \left\{ e = (v \to w) \mid \exists T \supseteq \{s\} : (v \xrightarrow{T} w) \in A_p \right\}$.

It is important to note that our Assumption 1 is too weak to guarantee that eventually the graph $A_p|s$ will ever exactly match the actual $\mathcal{G}^s$ in some round s. In fact, there might be a process q that does not have any incoming links from other processes, throughout the entire run of the algorithm. In that case, q cannot learn anything about the remaining network, i.e., A_q will permanently be the singleton graph.

The underlying idea of our consensus algorithm is to use flooding to forward the largest proposed value to everyone. However, as Assumption 1 does not guarantee bidirectional communication between every pair of processes, flooding is not sufficient: The largest proposal value could be known only to a single process that never has outgoing edges. Therefore, we let "privileged" processes, namely, the ones in a vertex-stable root component, try to impose their largest proposal values on the other processes. In order to do, so we use the well-known technique of locking a unique value. Processes only decide on their locked value once they are sure that every other process has locked this value as well. Since Assumption 1 guarantees that there will be one root component such that the processes in the root component can communicate their locked value to all other processes in the system they will eventually succeed.

Theorem 1. *Let r_{ST} be the first round where Assumption 1 holds. There is an algorithm that solves consensus by round $r_{ST} + 4D + 1$.*

5 Impossibilities and Lower Bounds

In this section, we will present a number of results that show that our basic Assumption 1, in particular, the existence of a stable window (of a certain minimal size) and the knowledge of an upper bound D on the causal network diameter, are crucial for making consensus solvable. Moreover, we will show that it is not unduly strong, as many problems considered in distributed systems in general (and dynamic networks in particular) remain unsolvable.

Although there is a strong bond between some of these problems and consensus in traditional settings, they are *not* implementable under our assumptions—basically, because there is no guarantee of (eventual) bidirectional communication.

Theorem 2. *Suppose that Assumption 1 is the only restriction on the adversary in our model. Then, neither* reliable broadcast*,* atomic broadcast*, nor* causal-order broadcast *can be implemented. Moreover, there is no algorithm that solves* counting*, k-*verification*, k-*token dissemination*,* all-to-all token dissemination*, and k-*committee election*.*

Theorem 3 (Knowledge of a Bound on the Network Causal Diameter). *Consider a system where Assumption 1 holds and suppose that processes do not know an upper bound D on the network causal diameter (and hence do not know n). Then, there is no deterministic algorithm that solves consensus.*

We now state a result that shows that it is necessary to have root components that are vertex stable long enough to flood the network. That is, w.r.t. Assumption 1, we need I to be in the order of D. To this end, we first introduce the following alternative Assumption 2, which requires a window of only D:

Assumption 2. *For any round r, there is exactly one root component $\mathcal{R}^r$ in $\mathcal{G}^r$. Moreover, there exists a D and an interval of rounds $I = [r_{ST}, r_{ST} + D]$, such that there is an I-vertex stable root component $\mathcal{R}^I$, such that $D^I \leqslant D$.*

In order to show that Assumption 2 is necessary, we further shorten the interval: Some process could possibly not be reached within $D - 1$ rounds, but would be reached if the interval was D rounds. Processes could hence *withold* information from each other, which causes consensus to be impossible [16].

Theorem 4. *Assume that Assumption 2 is not guaranteed in a system. Then consensus is impossible.*

6 Conclusion

We introduced a novel framework for modeling dynamic networks with directed communication links, and introduced a weak connectivity assumption that makes consensus solvable. Without such assumptions, consensus is trivially impossible in such systems as some processes can withhold their input values until a wrong decision has been made. We presented an algorithm that achieves consensus under this assumption, and showed several impossibility results and lower bounds that reveal that our algorithm is asymptotically optimal.

References

1. Goiser, A., Khattab, S., Fassl, G., Schmid, U.: A new robust interference reduction scheme for low complexity direct-sequence spread-spectrum receivers: Performance. In: Proceedings 3rd International IEEE Conference on Communication Theory, Reliability, and Quality of Service (CTRQ 2010), pp. 15–21 (June 2010)
2. Ware, C., Judge, J., Chicharo, J., Dutkiewicz, E.: Unfairness and capture behaviour in 802.11 adhoc networks. In: 2000 IEEE International Conference on Communications, ICC 2000. Global Convergence Through Communications (2000)
3. Kuhn, F., Oshman, R., Moses, Y.: Coordinated consensus in dynamic networks. In: Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2011. ACM (2011)
4. Biely, M., Robinson, P., Schmid, U.: Agreement in directed dynamic networks. CoRR abs/1204.0641 (2012)
5. Biely, M., Robinson, P., Schmid, U.: Solving k-set agreement with stable skeleton graphs. In: IPDPS Workshops, pp. 1488–1495 (2011)
6. Afek, Y., Gafni, E., Rosen, A.: The slide mechanism with applications in dynamic networks. In: ACM PODC, pp. 35–46 (1992)
7. Awerbuch, B., Patt-Shamir, B., Peleg, D., Saks, M.E.: Adapting to asynchronous dynamic networks. In: STOC 1992, pp. 557–570 (1992)

8. Harary, F., Gupta, G.: Dynamic graph models. Mathematical and Computer Modelling 25, 79–87 (1997)
9. Kuhn, F., Oshman, R.: Dynamic networks: Models and algorithms. SIGACT News 42(1), 82–96 (2011)
10. 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)
11. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: ACM STOC, pp. 513–522 (2010)
12. Santoro, N., Widmayer, P.: Time is Not a Healer. In: Cori, R., Monien, B. (eds.) STACS 1989. LNCS, vol. 349, pp. 304–313. Springer, Heidelberg (1989)
13. Charron-Bost, B., Schiper, A.: The Heard-Of model: computing in distributed systems with benign faults. Distributed Computing 22(1), 49–71 (2009)
14. Biely, M., Charron-Bost, B., Gaillard, A., Hutle, M., Schiper, A., Widder, J.: Tolerating corrupted communication. In: Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC 2007), pp. 244–253. ACM (2007)
15. Gafni, E.: Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony. In: Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, pp. 143–152. ACM Press (1998)
16. Schmid, U., Weiss, B., Keidar, I.: Impossibility results and lower bounds for consensus under link failures. SIAM Journal on Computing 38(5), 1912–1951 (2009)
17. Biely, M., Schmid, U., Weiss, B.: Synchronous consensus under hybrid process and link failures. Theoretical Computer Science 412(40), 5602–5630 (2011)
18. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics ans Applications, Philadelphia (2000)
19. Elrad, T., Francez, N.: Decomposition of distributed programs into communication-closed layers. Science of Computer Programming 2(3), 155–173 (1982)
20. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)

Bounding Interference in Wireless Ad Hoc Networks with Nodes in Random Position*

Majid Khabbazian[1], Stephane Durocher[2], and Alireza Haghnegahdar[3]

[1] University of Winnipeg, Winnipeg, Canada
m.khabbazian@uwinnipeg.ca
[2] University of Manitoba, Winnipeg, Canada
durocher@cs.umanitoba.ca
[3] University of British Columbia, Vancouver, Canada
alirezah@ece.ubc.ca

Abstract. Given a set of positions for wireless nodes, the interference minimization problem is to assign a transmission radius (equivalently, a power level) to each node such that the resulting communication graph is connected, while minimizing the maximum interference. We consider the model introduced by von Rickenbach et al. (2005), in which each transmission range is represented by a ball and edges in the communication graph are symmetric. The problem is NP-complete in two dimensions (Buchin 2008) and no polynomial-time approximation algorithm is known. In this paper we show how to solve the problem efficiently in settings typical for wireless ad hoc networks. We show that if node positions are represented by a set P of n points selected uniformly and independently at random over a d-dimensional rectangular region, for any fixed d, then the topology given by the closure of the Euclidean minimum spanning tree of P has maximum interference $O(\log n)$ with high probability. We extend this bound to a general class of communication graphs over a broad set of probability distributions. We present a local algorithm that constructs a graph from this class; this is the first local algorithm to provide an upper bound on the expected maximum interference. Finally, we analyze an empirical evaluation of our algorithm by simulation.

1 Introduction

1.1 Motivation

Establishing connectivity in a wireless network can be a complex task for which various (sometimes conflicting) objectives must be optimized. To permit a packet to be routed from any origin node to any destination node in the network, the corresponding communication graph must be connected. In addition to requiring connectivity, various properties can be imposed on the network, including low

* This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). The full version of this paper was posted online by the authors in November 2011 [15]. Proofs omitted here due to space constraints can found in the full version.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 85–98, 2012.

power consumption [20,27], bounded average traffic load [10,12], small average hop distance between sender-receiver pairs [1], low dilation (t-spanner) [1,3,6,7, 14,21,25], and minimal interference; this latter objective, minimizing interference (and, consequently, minimizing the required bandwidth), is the focus of much recent research [1,2,5,9,11,17–19,22–24,27–31] and of this paper.

We adopt the interference model introduced by von Rickenbach et al. [30] (see Section 1.2). We model transmission in a wireless network by assigning to each wireless node p a radius of transmission $r(p)$, such that every node within distance $r(p)$ of p can receive a transmission from p, whereas no node a greater distance from p can. The interference at node p is the number of nodes that have p within their respective radii of transmission. Given a set of wireless nodes whose positions are represented by a set of points P, we consider the problem of identifying a connected network on P that minimizes the maximum interference. The problem of constructing the network is equivalent to that of assigning a transmission radius to each node; once the transmission radius of each node is fixed, the corresponding communication graph and its associated maximum interference are also determined. Conversely, once a graph is fixed, each node's transmission radius is determined by the distance to its furthest neighbour.

Given a set of points P in the plane, finding a connected graph on P that minimizes the maximum interference is NP-complete [5]. A polynomial-time algorithm exists that returns a solution with maximum interference $O(\sqrt{n})$, where $n = |P|$ [11]. Even in one dimension, for every n there exists a set of n points P such that any graph on P has maximum interference $\Omega(\sqrt{n})$ [30]. All such known examples involve specific constructions (i.e., exponential chains). We are interested in investigating a more realistic class of wireless networks: those whose node positions observe common random distributions that better model actual wireless ad hoc networks.

When nodes are positioned on a line (often called the *highway model*), a simple heuristic is to assign to each node a radius of transmission that corresponds to the maximum of the distances to its respective nearest neighbours to the left and right. In the worst case, such a strategy can result in $\Theta(n)$ maximum interference when an optimal solution has only $\Theta(\sqrt{n})$ maximum interference [30]. Recently, Kranakis et al. [19] showed that if n nodes are positioned uniformly at random on an interval, then the maximum interference provided by this heuristic is $\Theta(\sqrt{\log n})$ with high probability.

In this paper, we examine the corresponding problem in two and higher dimensions. We generalize the nearest-neighbour path used in the highway model to the Euclidean minimum spanning tree (MST), and show that with high probability, the maximum interference of the MST of a set of n points selected uniformly at random over a d-dimensional region $[0, 1]^d$ is $O(\log n)$, for any fixed $d \geq 1$. Our techniques differ significantly from those used by Kranakis et al. to achieve their results in one dimension. As we show in Section 3, our results also apply to a broad class of random distributions, denoted $\mathcal{D}$, that includes both the uniform random distribution and realistic distributions for modelling random motion in

mobile wireless networks, as well as to a large class of connected spanning graphs that includes the MST.

In Section 3.4 we present a local algorithm that constructs a topology whose maximum interference is $O(\log n)$ with high probability when node positions are selected according to a distribution in $\mathcal{D}$. Previous local algorithms for topology control (e.g., the cone-based local algorithm (CBTC) [20]) attempt to reduce transmission radii (i.e., power consumption), but not necessarily the maximum interference. Although reducing transmission radii at many nodes is often necessary to reduce the maximum interference, the two objectives differ; specifically, some nodes may require large transmission radii to minimize the maximum interference. Ours is the first local algorithm to provide a non-trivial upper bound on maximum interference. Our algorithm can be applied to any existing topology to refine it and further reduce its maximum interference. Consequently, our solution can be used either independently, or paired with another topology control strategy. Finally, we discuss an empirical evaluation of our algorithm with a suite of simulation results in Section 4.

1.2 Model and Definitions

We represent the position of a wireless node as a point in Euclidean space, $\mathbb{R}^d$, for some fixed $d \geq 1$. For simplicity, we refer to each node by its corresponding point. Similarly, we represent a wireless network by its communication graph, a geometric graph whose vertices are a set of points $P \subseteq \mathbb{R}^d$. Given a (simple and undirected) graph G, we employ standard graph-theoretic notation, where $V(G)$ denotes the vertex set of G and $E(G)$ denotes its edge set. We say vertices u and v are *k-hop neighbours* if there is a simple path of length k from u to v in G. When $k = 1$ we say u and v are neighbours.

We assume a uniform range of communication for each node and consider bidirectional communication links, each of which is represented by an undirected graph edge connecting two nodes. Specifically, each node p has some *radius of transmission*, denoted by the function $r : P \to \mathbb{R}^+$, such that a node q receives a transmission from p if and only if $\text{dist}(p, q) \leq r(p)$, where $\text{dist}(p, q) = \|p - q\|_2$ denotes the Euclidean distance between points p and q in $\mathbb{R}^d$. For simplicity, suppose each node has an infinite radius of reception, regardless of its radius of transmission.

Definition 1 (Communication Graph). *A graph G is a* communication graph *with respect to a point set $P \subseteq \mathbb{R}^d$ and a function $r : P \to \mathbb{R}^+$ if (i) $V(G) = P$, and*

$$\textit{(ii)}\ \forall\{p, q\} \subseteq V(G),\ \{p, q\} \in E(G) \Leftrightarrow \text{dist}(p, q) \leq \min\{r(p), r(q)\}. \quad (1)$$

Together, set P and function r uniquely determine the corresponding communication graph G. Alternatively, a communication graph can be defined as the closure of a given embedded graph. Specifically, if instead of being given P and r, we are given an arbitrary graph H embedded in $\mathbb{R}^d$, then the set P is trivially

determined by $V(H)$ and a transmission radius for each node $p \in V(H)$ can be assigned to satisfy (1) by

$$r(p) = \max_{q \in \mathrm{Adj}(p)} \mathrm{dist}(p, q), \tag{2}$$

where $\mathrm{Adj}(p) = \{q \mid \{q, p\} \in E(H)\}$ denotes the set of vertices adjacent to p in H. The communication graph determined by H is the unique edge-minimal supergraph of H that satisfies Definition 1. We denote this graph by H' and refer to it as the *closure* of graph H. Therefore, a communication graph G can be defined either as a function of a set of points P and an associated mapping of transmission radii $r : P \to \mathbb{R}^+$, or as the closure of a given embedded graph H (where $G = H'$).

Definition 2 (Interference). *Given a communication graph G, the* interference *at node p in $V(G)$ is*

$$\mathrm{inter}_G(p) = |\{q \mid q \in V(G) \setminus \{p\} \text{ and } \mathrm{dist}(q, p) \leq r(q)\}|$$

and the maximum interference *of G is* $\mathrm{inter}(G) = \max_{p \in V(G)} \mathrm{inter}_G(p)$.

In other words, the interference at node p, denoted $\mathrm{inter}_G(p)$, is the number of nodes q such that p lies within q's radius of transmission. This does not imply the existence of the edge $\{p, q\}$ in the corresponding communication graph; such an edges exists if and only if the relationship is reciprocal, i.e., q also lies within p's radius of transmission.

Given a point set P, let $\mathcal{G}(P)$ denote the set of connected communication graphs on P. Let $\mathrm{OPT}(P)$ denote the optimal maximum interference attainable over graphs in $\mathcal{G}(P)$. That is,

$$\mathrm{OPT}(P) = \min_{G \in \mathcal{G}(P)} \mathrm{inter}(G) = \min_{G \in \mathcal{G}(P)} \max_{p \in V(G)} \mathrm{inter}_G(p).$$

Thus, given a set of points P representing the positions of wireless nodes, the *interference minimization problem* is to find a connected communication graph G on P that spans P such that the maximum interference is minimized (i.e., its maximum interference is $\mathrm{OPT}(P)$). In this paper we examine the maximum interference of the communication graph determined by the closure of $\mathrm{MST}(P)$, where $\mathrm{MST}(P)$ denotes the Euclidean minimum spanning tree of the point set P. Our results apply with high probability, which refers to probability at least $1 - n^{-c}$, where $n = |P|$ denotes the number of networks nodes and $c \geq 1$ is fixed.

2 Related Work

Bidirectional Interference Model. In this paper we consider the bidirectional interference model (defined in Section 1.2). This model was introduced by von Rickenbach et al. [30], who gave a polynomial-time approximation algorithm that finds a solution with maximum interference $O(n^{1/4} \cdot \mathrm{OPT}(P))$ for any given set of

points P on a line, and a one-dimensional construction showing that $\mathrm{OPT}(P) \in \Omega(\sqrt{n})$ in the worst case, where $n = |P|$. Halldórsson and Tokuyama [11] gave a polynomial-time algorithm that returns a solution with maximum interference $O(\sqrt{n})$ for any given set of n points in the plane. Buchin [5] showed that finding an optimal solution (one whose maximum interference is exactly $\mathrm{OPT}(P)$) is NP-complete in the plane. Tan et al. [29] gave an $O(n^3 n^{O(\mathrm{OPT}(P))})$-time algorithm for finding an optimal solution for any given set of points P on a line. Kranakis et al. [19] showed that for any set of n points P selected uniformly at random from the unit interval, the maximum interference of the nearest-neighbour path ($\mathrm{MST}(P)'$) has maximum interference $\Theta(\sqrt{\log n})$ with high probability. Sharma et al. [28] consider heuristic solutions to the two-dimensional problem. Finally, recent results by Devroye and Morin [9] extend some of the results presented in this paper and answer a number of open questions definitively to show that with high probability, when P is a set of n points in $\mathbb{R}^d$ selected uniformly at random from $[0,1]^d$, $\mathrm{inter}(\mathrm{MST}(P)') \in \Theta((\log n)^{1/2})$, $\mathrm{OPT}(P) \in O((\log n)^{1/3})$, and $\mathrm{OPT}(P) \in \Omega((\log n)^{1/4})$.

Unidirectional Interference Model. If communication links are not bidirectional (i.e., edges are directed) and the communication graph is required to be strongly connected, then the worst-case maximum interference decreases. Under this model, von Rickenbach et al. [31] and Korman [17] give polynomial-time algorithms that return solutions with maximum interference $O(\log n)$ for any given set of points in the plane, and a one-dimensional construction showing that in the worst case $\mathrm{OPT}(P) \in \Omega(\log n)$.

Minimizing Average Interference. In addition to results that examine the problem of minimizing the maximum interference, some work has addressed the problem of minimizing the average interference, e.g., Tan et al. [29] and Moscibroda and Wattenhofer [24].

3 Bounds

3.1 Generalizing One-Dimensional Solutions

Before presenting our results on random sets of points, we begin with a brief discussion regarding the possibility of generalizing existing algorithms that provide approximate solutions for one-dimensional instances of the interference minimization problem (in an adversarial deterministic input setting).

Since the problem of identifying a graph that achieves the optimal (minimum) interference is NP-hard in two or more dimensions [5], it is natural to ask whether one can design a polynomial-time algorithm to return a good approximate solution. Although Rickenbach et al. [30] give a $\Theta(n^{1/4})$-approximate algorithm in one dimension [30], the current best polynomial-time algorithm in two (or more) dimensions by Halldórsson and Tokuyama [11] returns a solution whose maximum interference is $O(\sqrt{n})$; as noted by Halldórsson and Tokuyama, this algorithm is not known to guarantee any approximation factor better than the immediate bound of $O(\sqrt{n})$. The algorithm of Rickenbach et al. uses two strategies

for constructing respective communication graphs, and returns the graph with the lower maximum interference; an elegant argument that depends on Lemma 1 bounds the resulting worst-case maximum interference by $\Theta(n^{1/4} \cdot \mathrm{OPT}(P))$. The two strategies correspond roughly to a) $\mathrm{MST}(P)'$ and b) classifying every $\sqrt{n}$th node as a hub, joining each hub to its left and right neighbouring hubs to form a network backbone, and connecting each remaining node to its closest hub. The algorithm of Halldórsson and Tokuyama applies ϵ-nets, resulting in a strategy that is loosely analogous to a generalization of the hub strategy of Rickenbach et al. to higher dimensions. One might wonder whether the hybrid approach of Rickenbach et al. might be applicable in higher dimensions by returning $\mathrm{MST}(P)'$ or the communication graph constructed by the algorithm of Halldórsson and Tokuyama, whichever has lower maximum interference. To apply this idea directly would require generalizing the following property established by von Rickenbach et al. to higher dimensions:

Lemma 1 (von Rickenbach et al. [30] (2005)). *For any set of points $P \subseteq \mathbb{R}$,*

$$\mathrm{OPT}(P) \in \Omega\left(\sqrt{\mathrm{inter}(\mathrm{MST}(P)')}\right).$$

However, von Rickenbach et al. also show that for any n, there exists a set of n points $P \subseteq \mathbb{R}^2$ such that $\mathrm{OPT}(P) \in O(1)$ and $\mathrm{inter}(\mathrm{MST}(P)') \in \Theta(n)$, which implies that Lemma 1 does not hold in higher dimensions. Consequently, techniques such as those used by von Rickenbach et al. do not immediately generalize to higher dimensions.

3.2 Randomized Point Sets

Although using the hybrid approach of von Rickenbach et al. [30] directly may not be possible, Kranakis et al. [19] recently showed that if a set P of n points is selected uniformly at random from an interval, then the maximum interference of the communication graph determined by $\mathrm{MST}(P)'$ is $\Theta(\sqrt{\log n})$ with high probability. Throughout this section, we assume general position of points; specifically, we assume that the distance between each pair of nodes is unique.

We begin by introducing some definitions. An edge $\{p, q\} \in E(G)$ in a communication graph G is *primitive* if $\min\{r(p), r(q)\} = \mathrm{dist}(p, q)$. An edge $\{p, q\} \in E(G)$ in a communication graph G is *bridged* if there is a path joining p and q in G consisting of at most three edges, each of which is of length less than $\mathrm{dist}(p, q)$. Given a set of points P in $\mathbb{R}^d$, let $\mathcal{T}(P)$ denote the set of all communication graphs G with $V(G) = P$ such that no primitive edge in $E(G)$ is bridged.

Halldórsson and Tokuyama [11] and Maheshwari et al. [23] give respective centralized algorithms for constructing graphs G, each with interference $O(\log(d_{\max}(G)/d_{\min}(G)))$, where $d_{\max}(G)$ and $d_{\min}(G)$ are defined as in Theorem 1. As we show in Theorem 1, this bound holds for any graph G in the class $\mathcal{T}(P)$. In Section 3.4 we give a local algorithm for constructing a connected graph in $\mathcal{T}(P)$ on any given point set P.

Theorem 1. *Let P be a set of points in $\mathbb{R}^d$. For any graph $G \in \mathcal{T}(P)$,*

$$\text{inter}(G) \in O\left(\log\left(\frac{d_{\max}(G)}{d_{\min}(G)}\right)\right),$$

where $d_{\max}(G) = \max_{\{s,t\}\in E(G)} \text{dist}(s,t)$ and $d_{\min}(G) = \min_{\{s,t\}\in E(G)} \text{dist}(s,t)$.

The proof is omitted due to space constraints. In the next lemma we show that $\text{MST}(P)'$ is in $\mathcal{T}(P)$. Consequently, $\mathcal{T}(P)$ is always non-empty.

Lemma 2. *For any set of points $P \subseteq \mathbb{R}^d$, $\text{MST}(P)' \in \mathcal{T}(P)$.*

Proof. The transmission range of each node $p \in P$ is determined by the length of the longest edge adjacent to p in $\text{MST}(P)$. Suppose there is a primitive edge $\{p_1, p_2\} \in E(\text{MST}(P))$ that is bridged. Therefore, there is a path T from p_1 to p_2 in $\text{MST}(P)'$ that contains at most three edges, each of which is of length less than $\text{dist}(p_1, p_2)$. Removing the edge $\{p_1, p_2\}$ partitions $\text{MST}(P)$ into two connected components, where p_1 and p_2 are in different components. By definition, T contains an edge that spans the two components. The two components can be joined using this edge (of length less than $\text{dist}(p_1, p_2)$) to obtain a new spanning tree whose weight is less than that of $\text{MST}(P)$, deriving a contradiction. Therefore, no primitive edge $\{p_1, p_2\} \in \text{MST}(P)$ can be bridged, implying $\text{MST}(P)' \in \mathcal{T}(P)$. □

Theorem 1 implies that the interference of any graph G in $\mathcal{T}(P)$ is bounded asymptotically by the logarithm of the ratio of the longest and shortest edges in G. While this ratio can be arbitrarily large in the worst case, we show that the ratio is bounded for many typical distributions of points. Specifically, if the ratio is $O(n^c)$ for some constant c, then the maximum interference is $O(\log n)$.

Definition 3 ($\mathcal{D}$). *Let $\mathcal{D}$ denote the class of distributions over $[0,1]^d$ such that for any $D \in \mathcal{D}$ and any set P of $n \geq 2$ points selected independently at random according to D, the minimum distance between any two points in P is greater than n^{-c} with high probability, for some constant c (independent of n).*

Theorem 2. *For any integers $d \geq 1$ and $n \geq 2$, any distribution $D \in \mathcal{D}$, and any set P of n points, each of which is selected independently at random over $[0,1]^d$ according to distribution D, with high probability, for all graphs $G \in \mathcal{T}(P)$, $\text{inter}(G) \in O(\log n)$.*

Proof. Let $d_{\min}(G) = \min_{\{s,t\}\in E(G)} \text{dist}(s,t)$ and $d_{\max}(G) = \max_{\{s,t\}\in E(G)} \text{dist}(s,t)$. Since points are contained in $[0,1]^d$, $d_{\max}(G) \leq \sqrt{d}$. Points in P are distributed according to a distribution $D \in \mathcal{D}$. By Definition 3, with high probability, $d_{\min}(G) \geq n^{-c}$ for some constant c. Thus, with high probability, we have

$$\log\left(\frac{d_{\max}(G)}{d_{\min}(G)}\right) \leq \log\left(\frac{\sqrt{d}}{n^{-c}}\right). \tag{3}$$

The result follows from (3), Theorem 1, and the fact that $\log(n^c\sqrt{d}) \in O(\log n)$ when d and c are constant. □

Lemma 3. *Let D be a distribution with domain $[0,1]^d$, for which there is a constant c' such that for any point $x \in [0,1]^d$, we have $D(x) \leq c'$, where $D(x)$ denotes the probability density function of D at $x \in [0,1]^d$. Then $D \in \mathcal{D}$.*

The proof is omitted due to space constraints.

Corollary 1. *The uniform distribution with domain $[0,1]^d$ is in $\mathcal{D}$.*

By Corollary 1 and Theorem 2, we can conclude that if a set P of $n \geq 2$ points is distributed uniformly in $[0,1]^d$, then with high probability, any communication graph in $G \in \mathcal{T}(P)$ will have maximum interference $O(\log n)$. This is expressed formally in the following corollary:

Corollary 2. *Choose any integers $d \geq 1$ and $n \geq 2$. Let P be a set of n points, each of which is selected independently and uniformly at random over $[0,1]^d$. With high probability, for all graphs $G \in \mathcal{T}(P)$, $\text{inter}(G) \in O(\log n)$.*

3.3 Mobility

Our results apply to the setting of mobility (e.g., mobile ad hoc wireless networks). Each node in a mobile network must periodically exchange information with its neighbours to update its local data storing positions and transmission radii of nodes within its local neighbourhood. The distribution of mobile nodes depends on the mobility model, which is not necessarily uniform. For example, when the network is distributed over a disc or a box-shaped region, the probability distribution associated with the random waypoint model achieves its maximum at the centre of the region, whereas the probability of finding a node close to the region's boundary approaches zero [12]. Since the maximum value of the probability distribution associated with the random waypoint model is constant [12], by Lemma 3 and Theorem 2, we can conclude that at any point in time, the maximum interference of the network is $O(\log n)$ with high probability. In general, this holds for any random mobility model whose corresponding probability distribution has a constant maximum value.

3.4 Local Algorithm

As discussed in Section 1.1, existing local algorithms for topology control attempt to reduce transmission radii, but not necessarily the maximum interference. By Lemma 2 and Theorem 2, if P is a set of n points selected according to a distribution in $\mathcal{D}$, then with high probability $\text{inter}(\text{MST}(P)') \in O(\log n)$. Unfortunately, a minimum spanning tree cannot be generated using only local information [16]. Thus, an interesting question is whether each node can assign itself a transmission radius using only local information such that the resulting communication graph belongs to $\mathcal{T}(P)$ while remaining connected. We answer this question affirmatively and present the first local algorithm (LocalRadiusReduction), that assigns a transmission radius to each node such that if the initial communication graph $G_{\max}$ is connected, then the resulting communication graph is a connected spanning subgraph of $G_{\max}$ that belongs to $\mathcal{T}(P)$. Consequently,

the resulting topology has maximum interference $O(\log n)$ with high probability when nodes are selected according to any distribution in $\mathcal{D}$. Our algorithm can be applied to any existing topology to refine it and further reduce its maximum interference. Thus, our solution can be used either independently, or paired with another topology control strategy. The algorithm consists of three phases, which we now describe.

Let P be a set of $n \geq 2$ points in $\mathbb{R}^d$ and let $r_{\max} : P \to \mathbb{R}^+$ be a function that returns the maximum transmission radius allowable at each node. Let $G_{\max}$ denote the communication graph determined by P and $r_{\max}$. Suppose $G_{\max}$ is connected. Algorithm LocalRadiusReduction assumes that each node is initially aware of its maximum transmission radius, its spatial coordinates, and its unique identifier.

The algorithm begins with a local data acquisition phase, during which every node broadcasts its identity, maximum transmission radius, and coordinates in a node data message. Each message also specifies whether the data is associated with the sender or whether it is forwarded from a neighbour. Every node records the node data it receives and retransmits those messages that were not previously forwarded. Upon completing this phase, each node is aware of the corresponding data for all nodes within its 2-hop neighbourhood. The algorithm then proceeds to an asynchronous transmission radius reduction phase.

Consider a node u and let f denote its furthest neighbour. If u and f are bridged in $G_{\max}$, then u reduces its transmission radius to correspond to that of its next-furthest neighbour f', where $\text{dist}(u, f') < \text{dist}(u, f)$. This process iterates until u is not bridged with its furthest neighbour within its reduced transmission radius. We formalize the local transmission radius reduction algorithm in the pseudocode in Table 1 that computes the new transmission radius $r'(u)$ at node u.

Table 1. pseudocode for Algorithm LocalRadiusReduction(u)

```
1  radiusReductionComplete ← false
2  r'(u) ← r_max(u)
3  f ← u
4  for each v ∈ Adj(u)
5     if dist(u, v) > dist(u, f)
6        f ← v // furthest neighbour
7  while ¬radiusReductionComplete
8     radiusModified ← false
9     if Bridged(u, f)
10       radiusModified ← true
11       f ← u // identify next neighbour within distance r'(u)
12       for each v ∈ Adj(u)
13          if dist(u, v) < r'(u) and dist(u, v) > dist(u, f)
14             f ← v
15       r'(u) ← dist(u, f)
16    radiusReductionComplete ← ¬radiusModified
17 return r'(u)
```

Algorithm LocalRadiusReduction is 2-local. Since transmission radii are decreased monotonically (and never increased), the while loop iterates $O(\Delta)$ times, where Δ denotes the maximum vertex degree in $G_{\max}$. Since each call to the subroutine Bridged terminates in $O(\Delta^2)$ time, each node determines its reduced transmission radius $r'(u)$ in $O(\Delta^3)$ time.

After completing the transmission radius reduction phase, the algorithm concludes with one final adjustment in the transmission radius to remove asymmetric edges. In this third and final phase, each node u broadcasts its reduced transmission radius $r'(u)$. Consider the set of nodes $\{v_1, \ldots, v_k\} \subseteq \mathrm{Adj}(u)$ such that $\mathrm{dist}(u, v_i) = r'(u)$ for all i (when points are in general position, $k = 1$, and there is a unique such node v_1). If $r'(v_i) < r'(u)$ for all i, then u can reduce its transmission radius to that of its furthest neighbour with which bidirectional communication is possible. Specifically,

$$r'(u) \leftarrow \max_{\substack{v \in \mathrm{Adj}(u) \\ \mathrm{dist}(u,v) \leq \min\{r'(u), r'(v)\}}} \mathrm{dist}(u, v). \tag{4}$$

The value of $r'(u)$ as defined in (4) is straightforward to compute in $O(\Delta)$ time.

Lemma 4. *The communication graph constructed by Algorithm* LocalRadiusReduction *is in* $\mathcal{T}(P)$ *and is connected if the initial communication graph* $G_{\max}$ *is connected.*

The proof is omitted due to space constraints. More generally, since transmission radii are only decreased, it can be shown that $G_{\min}$ and $G_{\max}$ have the same number of connected components by applying Lemma 4 on every connected component of $G_{\max}$.

4 Simulation

We simulated Algorithm LocalRadiusReduction to evaluate its performance in static and mobile wireless networks. In both settings, each node collects the list of its 2-hop neighbours in two rounds, applies the algorithm to reduce its transmission radius, and then broadcasts its computed transmission radius so neighbouring nodes can eliminate asymmetric edges and possibly further reduce their transmission radii. By the end of this stage, all asymmetric edges are removed and no new asymmetric edges are generated. Consequently, a node need not broadcast its transmission radius again after it has been further reduced.

We applied two mobility models to simulate mobile networks: random walk and random waypoint [13]. In both models each node's initial position is a point selected uniformly at random over the simulation region. In the random walk model, each node selects a new speed and direction uniformly at random over $[v_{\min}, v_{\max}]$ and $[0, 2\pi)$, respectively, at regular intervals. When a node encounters the simulation region's boundary, its direction is reversed (a rotation of π) to remain within the simulation region with the same speed. In the random waypoint model, each node moves along a straight trajectory with constant speed

toward a destination point selected uniformly at random over $[v_{\min}, v_{\max}]$ and the simulation region, respectively. Upon reaching its destination, the node stops for a random pause time, after which it selects a new random destination and speed, and the process repeats.

We set the simulation region's dimensions to 1000 metres × 1000 metres. For both static and dynamic networks, we varied the number of nodes n from 50 to 1000 in increments of 50. We fixed the maximum transmission radius $r_{\max}$ for each network to 100, 200, or 300 metres. To compute the average maximum interference for static networks, for each n and $r_{\max}$ we generated 100,000 static networks, each with n nodes and maximum transmission radius $r_{\max}$, distributed uniformly at random in the simulation region. To compute the average maximum interference for mobile networks, for each n and $r_{\max}$ we generated 100,000 snapshots for each mobility model, each with n nodes and maximum transmission radius $r_{\max}$. We set the speed interval to $[0.2, 10]$ metres per second, and the pause time interval to $[0, 10]$ seconds (in the waypoint model). A snapshot of the network was recorded once every second over a simulation of 100,000 seconds.

We compared the average maximum interference of the topology constructed by the algorithm LocalRadiusReduction against the corresponding average maximum interference achieved respectively by two local topology control algorithms: i) the local computation of the intersection of the Gabriel graph and the unit disc graph (with unit radius $r_{\max}$) [4], and ii) the cone-based local topology control (CBTC) algorithm [20]. In addition, we evaluated the maximum interference achieved when each node uses a fixed radius of communication, i.e., the communication graph is a unit disc graph of radius $r_{\max}$ (100, 200, or 300 metres, respectively). See the full version [15] for figures displaying simulation results.

As shown, the average maximum interference of unit disc graph topologies increases linearly with n. Many of the unit disc graphs generated were disconnected when the transmission radius was set to 100 metres for small n. Since we require connectivity, we only considered values of n and $r_{\max}$ for which at least half of the networks generated were connected. When $r_{\max} = 100$ metres, a higher average maximum interference was measured at $n = 300$ than at $n = 400$. This is because many networks generated for $n = 300$ were discarded due to being disconnected. Consequently, the density of networks simulated for $n = 300$ was higher than the average density of a random network with $n = 300$ nodes, resulting in higher maximum interference.

Although both the local Gabriel and CBTC algorithms performed significantly better than the unit disc graphs, the lowest average maximum interference was achieved by the LocalRadiusReduction algorithm. Note that the LocalRadiusReduction algorithm reduces the maximum interference to $O(\log n)$ with high probability, irrespective of the initial maximum transmission radius $r_{\max}$.

Simulation results obtained using the random walk model closely match those obtained on a static network because the distribution of nodes at any time during a random walk is nearly uniform [8]. The average maximum interference increases slightly but remains logarithmic when the random waypoint model is used. The spatial distribution of nodes moving according to a random waypoint model

is not uniform, and is maximized at the centre of the simulation region [12]. Consequently, the density of nodes is high near the centre, resulting in greater interference at these nodes.

Finally, we evaluated the algorithm LocalRadiusReduction using actual mobility trace data of Piorkowski et al. [26], consisting of GPS coordinates for trajectories of 537 taxi vehicles recorded over one month in 2008, driving throughout the San Fransisco Bay area. We selected the 500 largest traces, each of which has over 8000 sample points. To implement our algorithm, we selected n taxis among the 500 uniformly at random, ranging from $n = 50$ to $n = 500$ in increments of 50. The resulting average maximum interference is similar to that measured in our simulation results.

5 Discussion

Using Algorithm LocalRadiusReduction, each node determines its transmission radius as a function of its 2-hop neighbourhood. Alternatively, suppose each node could select its transmission radius at random using a suitable distribution over $[d_{\min}(G), d_{\max}(G)]$. Can such a strategy for assigning transmission radii ensure connectivity and low maximum interference with high probability? Similarly, additional topologies and local algorithms for constructing them might achieve $O(\log n)$ expected maximum interference. For example, our experimental results suggest that both the Gabriel graph and CBTC local topology control algorithms may provide $O(\log n)$ expected maximum interference. Since neither the Gabriel graph nor the CBTC topology of a set of points P is in $\mathcal{T}(P)$ in general, whether these bounds hold remains to be proved.

As mentioned in Section 2, multiple open questions related to interference on random sets of points were resolved recently by Devroye and Morin [9]. Several questions remain open related to the algorithmic problem of finding an *optimal* solution (one whose maximum interference is exactly $\mathrm{OPT}(P)$) when node positions may be selected adversarially. The complexity of the interference minimization in one dimension remains open; at present, it is unknown whether the problem is polynomial-time solvable or NP-hard [29]. While the problem is known to be NP-complete in two dimensions [5], no polynomial-time approximation algorithm nor any inapproximability hardness results are known.

Acknowledgements. Stephane Durocher thanks Csaba Tóth for insightful discussions related to the interference minimization problem in one dimension.

References

1. Benkert, M., Gudmundsson, J., Haverkort, H., Wolff, A.: Constructing minimum-interference networks. Comp. Geom.: Theory & App. 40(3), 179–194 (2008)
2. Bilò, D., Proietti, G.: On the complexity of minimizing interference in ad-hoc and sensor networks. Theor. Comp. Sci. 402(1), 42–55 (2008)
3. Bose, P., Gudmundsson, J., Smid, M.: Constructing plane spanners of bounded degree and low weight. Algorithmica 42(3-4), 249–264 (2005)

4. Bose, P., Morin, P., Stojmenović, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. Wireless Net. 7(6), 609–616 (2001)
5. Buchin, K.: Minimizing the maximum interference is hard. CoRR, abs/0802.2134 (2008)
6. Burkhart, M., von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Does topology control reduce interference. In: Proc. ACM MobiHoc, pp. 9–19 (2004)
7. Damian, M., Pandit, S., Pemmaraju, S.V.: Local approximation schemes for topology control. In: Proc. ACM PODC, pp. 208–218 (2006)
8. Das Sarma, A., Nanongkai, D., Pandurangan, G.: Fast distributed random walks. In: Proc. ACM PODC, pp. 161–170 (2009)
9. Devroye, L., Morin, P.: A note on interference in random point sets. CoRR, 1202.5945 (2012)
10. Durocher, S., Kranakis, E., Krizanc, D., Narayanan, L.: Balancing traffic load using one-turn rectilinear routing. J. Interconn. Net. 10(1-2), 93–120 (2009)
11. Halldórsson, M.M., Tokuyama, T.: Minimizing interference of a wireless ad-hoc network in a plane. Theor. Comp. Sci. 402(1), 29–42 (2008)
12. Hyytiä, E., Lassila, P., Virtamo, J.: Spatial node distribution of the random waypoint mobility model with applications. IEEE Trans. Mob. Comp. 6(5), 680–694 (2006)
13. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: Imielinski, T., Korth, H. (eds.) Mobile Computing, vol. 353, Kluwer Academic Publishers (1996)
14. Kanj, I., Perković, L., Xia, G.: Computing Lightweight Spanners Locally. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 365–378. Springer, Heidelberg (2008)
15. Khabbazian, M., Durocher, S., Haghnegahdar, A.: Bounding interference in wireless ad hoc networks with nodes in random position. CoRR, 1111.6689 (2011)
16. Khan, M., Pandurangan, G., Anil Kumar, V.S.: Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. IEEE Trans. Parallel & Dist. Sys. 20(1), 124–139 (2009)
17. Korman, M.: Minimizing Interference in Ad-Hoc Networks with Bounded Communication Radius. In: Asano, T., Nakano, S.-i., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 80–89. Springer, Heidelberg (2011)
18. Kowalski, D.R., Rokicki, M.A.: Connectivity Problem in Wireless Networks. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 344–358. Springer, Heidelberg (2010)
19. Kranakis, E., Krizanc, D., Morin, P., Narayanan, L., Stacho, L.: A tight bound on the maximum interference of random sensors in the highway model. CoRR, abs/1007.2120 (2010)
20. Li, L., Halpern, J.Y., Bahl, P., Wang, Y.-M., Wattenhofer, R.: A cone-based distributed topology-control algorithm for wireless multi-hop networks. IEEE/ACM Trans. Net. 13(1), 147–159 (2005)
21. Li, X.-Y., Calinescu, G., Wan, P.-J.: Distributed construction of a planar spanner and routing for ad hoc wireless networks. In: Proc. IEEE INFOCOM, pp. 1268–1277 (2002)
22. Locher, T., von Rickenbach, P., Wattenhofer, R.: Sensor Networks Continue to Puzzle: Selected Open Problems. In: Rao, S., Chatterjee, M., Jayanti, P., Murthy, C.S.R., Saha, S.K. (eds.) ICDCN 2008. LNCS, vol. 4904, pp. 25–38. Springer, Heidelberg (2008)

23. Maheshwari, A., Smid, M., Zeh, N.: Low-interference networks in metric spaces with bounded doubling dimension. Information Processing Letters 111(23-24), 1120–1123 (2011)
24. Moscibroda, T., Wattenhofer, R.: Minimizing interference in ad hoc and sensor networks. In: Proc. ACM DIALM-POMC, pp. 24–33 (2005)
25. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press (2007)
26. Piorkowski, M., Sarafijanovic-Djukic, N., Grossglauser, M.: CRAWDAD data set epfl/mobility (v. February 24, 2009), http://crawdad.cs.dartmouth.edu/epfl/mobility
27. Santi, P.: Topology control in wireless ad hoc and sensor networks. ACM Comp. Surv. 37(2), 164–194 (2005)
28. Sharma, A.K., Thakral, N., Udgata, S.K., Pujari, A.K.: Heuristics for Minimizing Interference in Sensor Networks. In: Garg, V., Wattenhofer, R., Kothapalli, K. (eds.) ICDCN 2009. LNCS, vol. 5408, pp. 49–54. Springer, Heidelberg (2008)
29. Tan, H., Lou, T., Lau, F.C.M., Wang, Y., Chen, S.: Minimizing Interference for the Highway Model in Wireless Ad-Hoc and Sensor Networks. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 520–532. Springer, Heidelberg (2011)
30. von Rickenbach, P., Schmid, S., Wattenhofer, R., Zollinger, A.: A robust interference model for wireless ad hoc networks. In: Proc. IEEE IPDPS, pp. 1–8 (2005)
31. von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Algorithmic models of interference in wireless ad hoc and sensor networks. IEEE/ACM Trans. Net. 17(1), 172–185 (2009)

Strong Connectivity of Sensor Networks with Double Antennae

Mohsen Eftekhari Hesari[1], Evangelos Kranakis[2,⋆], Fraser MacQuarie[3], Oscar Morales-Ponce[4,⋆⋆], and Lata Narayanan[5,⋆⋆⋆]

[1] Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Quebec
m_eftek@encs.concordia.ca
[2] School of Computer Science, Carleton University, Ottawa, Canada
kranakis@scs.carleton.ca
[3] School of Computer Science, Carleton University, Ottawa, Canada
frasermacquarrie@gmail.com
[4] School of Computer Science, Carleton University, Ottawa, Canada
omponce@connect.carleton.ca
[5] Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Quebec
lata@cs.concordia.ca

Abstract. Inspired by the well-known Dipole and Yagi antennae we introduce and study a new theoretical model of directional antennae that we call *double antennae*. Given a set P of n sensors in the plane equipped with double antennae of angle ϕ and with dipole-like and Yagi-like antenna propagation patterns, we study the *connectivity* and *stretch factor* problems, namely finding the minimum range such that double antennae of that range can be oriented so as to guarantee strong connectivity or stretch factor of the resulting network. We introduce the new concepts of $(2,\phi)$-connectivity and ϕ-angular range $r_\phi(P)$ and use it to characterize the optimality of our algorithms. We prove that $r_\phi(P)$ is a lower bound on the range required for strong connectivity and show how to compute $r_\phi(P)$ in time polynomial in n. We give algorithms for orienting the antennae so as to attain strong connectivity using optimal range when $\phi \geq 2\pi/3$, and algorithms approximating the range for $\phi \geq \pi/2$. For $\phi < \pi/3$, we show that the problem is NP-complete to approximate within a factor $\sqrt{3}$. For $\phi \geq \pi/2$, we give an algorithm to orient the antennae so that the resulting network has a stretch factor of at most 4 compared to the underlying unit disk graph.

Keywords: Connectivity, Double Antenna, Range, Stretch Factor, Unit Disk Graph.

1 Introduction

Directional antennae are versatile transceivers which are widely used in wireless communication. With proper design they are known to improve overall energy consump-

⋆ Supported in part by NSERC and MITACS grants.
⋆⋆ Supported by MITACS Postdoctoral Fellowship.
⋆⋆⋆ Supported in part by NSERC grants.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 99–110, 2012.

tion [12], enhance network capacity [9,15], improve topology control [8], and offer the potential for mitigating various security threats [10], just to mention a few applications. The motivation for our present study comes from the work in [2] which introduced the network connectivity problem for directional sensors and provided several algorithms for analyzing angle-range tradeoffs.

Dipole antennae (or *dipoles, for short*) are well-known basic antennae that are commonly used in radio communication. At their simplest, they consist of two straight collinear conductors of equal length separated by a small gap. Moreover, the radiation pattern for such antennae–indicating the strength of the signal in a given direction–in the *xy*-plane is usually depicted by two equal size closed curves known as *lobes*. Figure 1 illustrates the variability of the strength of the signal depending on the direction of the beam (see [14]). When the two lobes are not identical with the apex of one of the two lobes being closer to the origin than the other, the resulting antenna radiation pattern corresponds to a *Yagi antenna*, thus indicating that the antenna's transmission range is longer in one direction versus its opposite.

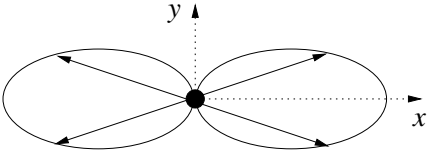


Fig. 1. Radiation pattern of a dipole antenna in the *xy*-plane

Motivated by the above, we introduce the following theoretical model of *Dipole-like* and *Yagi-like* antennae which we refer to as *double* antennae. These two concepts are captured in the following two geometric definitions.

Definition 1. *A* (ϕ, r)-double antenna *is an antenna with beamwidth or angle* ϕ *and radius r which can send and receive from either of the two sectors called beams depicted in Figure 2.*

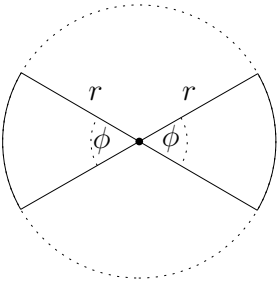


Fig. 2. Double antenna with beamwidth ϕ and range r

Definition 2. *More generally, a* (ϕ, r_1, r_2)-double *antenna is a double antenna with the range of one beam equal to* r_1 *and the range of the opposite beam* r_2.

Clearly, a (ϕ, r_1, r_1)-double antenna is also a $(\phi, \min\{r_1, r_2\})$-double antenna. Unless otherwise specified, in this paper, a double antenna refers to a (ϕ, r)-double antenna.

In this paper we are interested in the following two antenna orientation problems:

Problem 1 (Connectivity Problem with Double Antennae). Given a set P of n sensors in the plane each equipped with one double antenna with beamwidth $\phi \leq \pi$, determine the minimum antenna range, denoted by $\hat{r}_\phi(P)$, so that there exists an orientation of the antennae that induces a strongly connected transmission graph.

It is worth noting that for a sufficiently small angle ϕ, the problem is equivalent to the well-known bottleneck traveling salesman problem (BTSP) or Hamiltonian cycle that minimizes the longest edge. Therefore, a trivial upper-bound on the antenna range for $\phi \leq \pi$ of three times the optimal range can be computed by finding a Hamiltonian cycle with edge length bounded by three times the longest edge of the MST [3] [Problem C35.2-4] since the longest edge of the MST is also a lower bound for the orientation problem for any angle ϕ. However, for the BTSP a better analysis given in [13] shows that a 2-approximation can be obtained in polynomial time. Essentially, they proved that the lower bound for the BTSP is at least the longest edge of the 2-connected graph G that minimizes the longest edge. Thus, the 2-approximation is obtained easily since the square of any 2-connected graph is Hamiltonian [7].

Closely related to the orientation problem for attaining connectivity is the orientation problem to achieve constant stretch factor:

Problem 2 (Stretch Factor Problem with Double Antennae). Given a set P of n sensors in the plane each equipped with one double antenna with beamwidth $\phi \leq \pi$, determine the minimum antenna range so that there exists an orientation of the antennae that induces a c-directional spanner, where c is a constant.

1.1 Notation

We denote the Euclidean distance between points u and v by with $d(u,v)$. Let $UDG(P;r)$ denote the geometric graph (or straight line graph) such that P is the set of vertices and an edge $\{u,v\}$ exists if and only if $d(u,v) \leq r$. If r is normalized to be equal to 1 we simply denote the graph by $UDG(P)$. Throughout this paper the acronym UDG stands for Unit Disk Graph and the acronym MST for Euclidean Minimum Spanning Tree. Let $N_G(u)$ denote the set of neighbors of u.

Throughout this paper we assume that points are in general position, i.e., there do not exist three points that are collinear. It is well-known that the vertices of any MST have degree at most six since the angle that a vertex forms with two consecutive neighbors is at least $\pi/3$. However, when a vertex forms an angle of $\pi/3$ with every two consecutive neighbors, implies that at least three points are collinear. Hence, we assume that the max degree of the MST is five.

1.2 Related Work

The antenna orientation problem has been studied extensively since the problem was introduced by Caragianis et al [2]. for the single-directional antenna model. They proved

that the connectivity problem for a single antenna (per sensor) is NP-complete with beamwidth less than $2\pi/3$ and gave an upper-bound on the range for a beamwidth greater than π which is tight when the angle is at least $8\pi/5$. In [6], the authors studied the connectivity problem when each sensor has k antennae. They proposed an algorithm for orienting the antennae so as to obtain a strongly connected graph with out-degree bounded by k and longest edge bounded by $2\sin(\frac{\pi}{k+1})$ times the optimal length required to attain connectivity. A useful survey of the connectivity problem is presented in [11].

In all the previous results, the lower bound on the antenna range was based on the longest edge of the MST. In this paper we will introduce the new concept of $(2,\phi)$-connectivity for a given angle ϕ so as to characterize the optimal lower bound required for connectivity.

The stretch factor problem for a single-directional antenna was studied for the first time in [4] for the particular cases of angles $\pi/2$ and $2\pi/3$. They proved that a range of 7 and 5 is always sufficient to create a 6-directional spanner and a 5-directional spanner (i.e., with stretch-factor 6 and 5), respectively. A more comprenhensive result is given in [1] where the authors gave an upper bound for all angles.

1.3 Results of the Paper

We study the antenna orientation problems for connectivity and stretch-factor in the double antenna model. In Section 2 we introduce the new concepts of $(2,\phi)$-connectivity and ϕ-angular range $r_\phi(P)$ so as to characterize the optimality of our algorithms. Furthermore, we prove that $r_\phi(P)$ is the lower bound for the connectivity problem and show how to compute $r_\phi(P)$ in polynomial time. We prove tight bounds on the optimal angle necessary to cover all neighbours of a node in a MST in Section 3. Our results for the connectivity problem, including an NP-completeness proof when the beamwidth $\phi < \pi/3 - \varepsilon$ and an optimal algorithm for the case $\phi > 2\pi/3$ are presented in Section 4. In Section 5 we give a linear time algorithm for the stretch factor problem that orients the antennae of beamwidth at least $\pi/2$ so as to obtain a 4-directional spanner. Finally, we conclude in Section 6 and present some open problems. Our main results, complexities, and resulting angle/range tradeoffs for n sensors in the plane are summarized in Table 1.

2 Lower Bounds

In this section we characterize the lower bounds for the orientation problem for connectivity with double antennae. Consider an antenna orientation for a given antenna beam width ϕ and optimal range $\hat{r}_\phi(P)$ on a set of points P that induces a strongly connected graph on P. Given $u \in P$, let $v \in P$ be a point not in one of the beams of u's antenna such that $d(u,v) \leq \hat{r}_\phi(P)$. The main observation is that a path between u and v must exist such that each edge in the path is of length at most $\hat{r}_\phi(P)$. This implies that the unit disk graph $UDG(P;\hat{r}_\phi(P)) \setminus \{\{u,v\}\}$ is connected. We use this observation to obtain a lower bound on $\hat{r}_\phi(P)$.

Table 1. Results for the double antenna connectivity and stretch-factor problems on n sensors in the plane and for antennae of beam width ϕ

Double Antenna Angle	Approximation Ratio	Complexity	Stretch Factor
$\frac{2\pi}{3} \leq \phi < \pi$	1	$O(n^2)$	-
$\frac{\pi}{2} \leq \phi \leq \frac{2\pi}{3}$	$\sqrt{3}$	$O(n \log n)$	-
$\frac{\pi}{2} \leq \phi < \pi$	$4\sin(\frac{\pi}{4} + \frac{\phi}{2})$	$O(n)$	4
$0 \leq \phi < \frac{\pi}{2}$	3	$O(n \log n)$	-
$\phi < \frac{\pi}{3} - \varepsilon$	$\sqrt{3} - \varepsilon$	NP-Complete	-

First we define a double antenna orientation "relative to" a given angle γ. Let γ be a given angle oriented with its right edge (in counterclockwise direction) on the axis as depicted in Figure 3; $u(\phi;\gamma)$ denotes the orientation of the double antenna of beamwidth ϕ at u starting from the right edge (in counterclockwise direction) of γ (see Figure 3). Given the graph $UDG(P;r)$, and an orientation γ, we define $E_\phi(P,r,u,\gamma)$ as the subset of edges incident to u which lie outside the two beams of $u(\phi;\gamma)$ (see Figure 3).

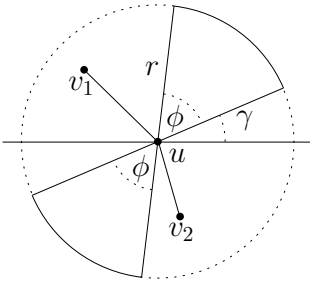


Fig. 3. $u(\phi;\gamma)$ denotes the double antenna with beam-width ϕ and orientation relative to γ. The edges $\{u,v_1\}$ and $\{u,v_2\}$ are in $E_\phi(P,r,u,\gamma)$.

Definition 3 (Set of Edges $E_\phi(P,r,u)$.). *Let $E_\phi(P,r,u)$ denote any set $E(P,r,u,\gamma)$, for $0 \leq \gamma \leq \pi$, such that the graph $UDG(P;r) \setminus E(P,r,u,\gamma)$ attains the minimum possible number of connected components.*

The following definition introduces the concept of $(2,\phi)$-connectivity of a UDG.

Definition 4 ($(2,\phi)$-connectivity.). *Let P be a set of points in the plane. We say that for a given radius r, the graph $UDG(P;r)$ is $(2,\phi)$-connected if for any vertex $u \in P$, $UDG(P;r) \setminus E_\phi(P,r,u)$ is connected.*

Observe that when ϕ is sufficiently small so that each antenna can only cover one vertex in its beams, the concept of $(2,\phi)$-connectivity is equivalent to the well-known concept of 2-connectivity.

Definition 5 (ϕ-angular radius.). *We define the ϕ-angular radius as the minimum radius, denoted by $r_\phi(P)$, such that $UDG(P; r_\phi(P))$ is $(2,\phi)-$connected.*

Now we will prove that the ϕ-angular radius is a lower bound for the orientation problem with double antennae. Due to space constraints the proof is omitted.

Theorem 1. *For any set P of points, $r_\phi(P) \leq \hat{r}_\phi(P)$.*

The following theorem gives a simple algorithm to compute $r_\phi(P)$ in polynomial time.

Theorem 2. *Given a set P of n points in the plane in general position and an angle $\phi \geq 0$, there is an algorithm that computes $r_\phi(P)$ in $O(n^2)$ time.*

Proof. Let T be an MST on P and let r be the length of the longest edge in T. Let $S \subseteq P$ be such that for each vertex $u \in S$ the graph $T \setminus E_\phi(P,r,u)$ is not connected. For $u \in S$, let $r_\phi(P,u)$ be the minimum range such that $UDG(P; r_\phi(P,u)) \setminus E_\phi(P, r_\phi(P,u))$ is connected. Clearly, $r_\phi(P) = \max_{u \in S}(r_\phi(P,u))$. We will determine $r_\phi(P,u)$ independently for every vertex $u \in S$.

Consider a vertex $u \in S$ and let $G = T$. We add to G the shortest edge $\{v,w\}$ that connects two distinct components of $G \setminus E_\phi(P,r,u)$. Update r to be the longest edge in G, and repeat the above procedure until $G \setminus E_\phi(P,r,u)$ is connected. Since the removal of the longest edge of G will disconnect the graph $G \setminus E_\phi(P,r,u)$, it follows that $r_\phi(u)$ equals the length of the longest edge in G.

It remains to analyze the complexity of the algorithm. Let $u_0, u_1, ..., u_{d_G(u)}$ be the neighbors of u. To find $E_\phi(P,r,u)$ we check for each neighbor u_i of u which orientation $G \setminus E_\phi(P,r,u,\angle(u_i u u_0))$ leaves the minimum number of components. We will show that the degree of u never exceeds five. Since T is an MST the max degree of u is five. However, new edges can increase the degree of u. Assume that $\{u,v\}$ is added to G. Since $\{u,v\}$ is the smallest edge that connects two distinct components of $G \setminus E_\phi(P,r,u)$ the angle that $\{u,v\}$ forms with the neighbors of u is at least $\pi/3$. Therefore, the max degree of u is bounded by five since P is in general position.

Next we show that the algorithm can be implemented in $O(n^2)$ time. First consider the Delaunay Triangulation on P and sort the edges in a list L. Such a construction takes $O(n\log(n))$ time [5]. Further, L can be computed in $O(n\log(n))$ time since the number of edges is linear on the number of vertices. It is well-known that the Grabriel Graph on P is a subgraph of the Delaunay Triangulation on P, i.e., each edge $\{v,w\} \in L$, $D(v; d(v,w)) \cap D(w; d(w,v)) = \emptyset$ (where $D(x;r)$ denotes the open disk centered at x with radius r). Therefore, for a given u we can compute the shortest edge $\{u,v\}$ connecting two components in $G \setminus E_\phi(P,r,u)$ in $O(n)$ time since $\{u,v\} \in L \cup G^k(N_G(u))$ where $G^k(N_G(u))$ represents the complete graph of $N_G(u)$ and $|N_G(u)| \leq 5$. The theorem follows, since each vertex has at most 5 connected components and $|S| \leq |P|$.

3 Covering Neighbors in MST with Double Antennae

Given an MST of a set of points P and a vertex $u \in V$ of degree $k \leq 5$ we will characterize the beamwidth required by an antenna at u to cover all the neighbors of u. Recall

that the MST on the set of points has maximum degree 5 and the angle between any two adjacent edges is at least $\pi/3$. Let $u_0, u_1, ..., u_{k-1}$ be the neighbors of u in G with corresponding angles $\alpha_i = \angle(u_i u u_{i+1})$ in counterclockwise order, where $k \leq 5$. We study separately the cases $k = 5, 4, 3, 2$ and in each case, we give the value of α, the minimum beamwidth of double antenna that is required to ensure that all neighbours of u fall within one of the beams of the antenna. Clearly any beamwdith $\phi \geq \alpha$ is always sufficient to cover all neighbours. Due to space constraints the proofs of Lemmas 2, 3 and 4 are omitted.

Lemma 1. *Let $k = 5$ and assume wlog that $\alpha_0 + \alpha_1$ is the smallest sum of two consecutive angles. Then, a double antenna of beamwidth $\alpha = \alpha_0 + \alpha_1$ is always sufficient and necessary to cover all five neighbors in the MST. Furthermore, $\alpha \in [\frac{2\pi}{3}, \frac{4\pi}{5}]$.*

Proof. Place the antenna as depicted in Figure 4 so that u_0 is on the edge of the antenna and u_1, u_2 are within the beam of the antenna. Since $\alpha_0, \alpha_1 \geq \pi/3$, we have $\alpha \geq 2\pi/3$. Therefore, the "dead" sectors of the antenna are of angle at most $\pi/3$. Since $\alpha_2, \alpha_4 \geq \pi/3$, neither u_3 nor u_4 can lie within the dead sectors of the antenna. Since three neighbors of u must be in the same side of the antenna beam, α is always necessary. Observe that $\frac{2\pi}{3} \leq \alpha \leq \frac{4\pi}{5}$ since all the angles are at least $\pi/3$.

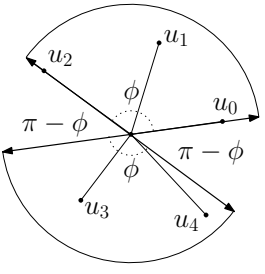


Fig. 4. Double antenna at u of degree 5

Lemma 2. *Let $k = 4$. Assume wlog that α_0 is the smallest angle and that $\alpha_1 \leq \alpha_3$. Then a double antenna of beamwidth α is always necessary and sufficient to cover all four neighbours in the MST, where $\alpha = \pi - \alpha_1$ if $\alpha_3 \geq \pi - \alpha_0$ and $\alpha = min(\alpha_2, \pi - \alpha_0)$ otherwise. Furthermore, $\alpha \in [\frac{\pi}{3}, \frac{2\pi}{3}]$.*

Lemma 3. *Let $k = 3$. Assume wlog that $\alpha_0 \leq \alpha_1 \leq \alpha_2$. Then a double antenna of beamwidth α is always sufficient and necessary to cover the three neighbors in the MST, where $\alpha = max\{\alpha_0, \pi - \alpha_1\}$. Furthermore, $\alpha \leq \frac{2\pi}{3}$.*

Lemma 4. *Let $k = 2$. Assume wlog that $\alpha_0 \leq \alpha_1$. Then a double antenna of beamwidth α is always sufficient and necessary to cover the two neighbors in the MST, where $\alpha = min\{\alpha_0, \pi - \alpha_0\}$. Furthermore, $\alpha \leq \frac{\pi}{2}$.*

4 Connectivity with Optimal Range

In this section we first show that the double antenna orientation problem is NP-complete for antenna angles less than $\frac{\pi}{3}$. In contrast, we will show that when the antenna beamwidth is sufficiently large, we can solve the orientation problem with optimal range. Due to space constraints the proofs of Theorems 3 and 4 are omitted.

Theorem 3. *For n sensors in the plane and $\phi < \frac{\pi}{3}$, it is NP-complete to approximate the optimal range $\hat{r}_\phi(P)$ to within a multiplicative factor of $\sqrt{3}$.*

Now we will show that when the antenna beamwidth is sufficiently large it is trivial to achieve optimal range as the next theorem shows.

Theorem 4. *Given an angle $\phi \geq \frac{4\pi}{5}$, there is an algorithm which for any set P of points in the plane in general position, orients double antennae of beamwidth ϕ using optimal antenna range so that the resulting graph is strongly connected.*

Next we will show how to achieve non-trivial optimal range with the use of the $(2,\phi)$-connectivity and ϕ-angular radius as a lower bound for an antenna beamwidth of at least $\frac{2\pi}{3}$.

Theorem 5. *Given an angle $\phi \geq \frac{2\pi}{3}$, there is an algorithm which for any set P of points in the plane in general position, orients double antennae of beamwidth ϕ using optimal antenna range so that the resulting graph is strongly connected. Furthermore, the algorithm can be implemented to run in $O(n^2)$ time.*

Proof. Let T be an Euclidean MST on P. Consider the set S of vertices $u \in T$ such that a double antenna of beamwidth ϕ cannot cover all the neighbors of u. From Lemmas 1-4, S consists only of vertices of degree five in T such that the angle that is formed with any three consecutive neighbors is greater than ϕ.

We will construct a strongly connected digraph such that every vertex in S has out-degree four and the angle that each vertex forms with two consecutive out-going edges is at least $\pi/3$. Furthermore, we will show that no new vertices of out-degree five will appear. Finally, all edges in the digraph will have length at most $r_\phi(P)$. Thus, the theorem follows from Lemmas 1-4 and Theorem 1.

Let $\overrightarrow{G}$ be the strongly connected directed graph obtained from T by replacing every edge in T by two opposing directed edges. Let G be the undirected graph of $\overrightarrow{G}$. We will include each vertex $u \in S$ in at least one cycle as follows:

Let u be any vertex in S and let $\{v,w\}$ be the shortest edge connecting two components of $G \setminus \{u\}$. Clearly, $d(v,w) \leq r_\phi(P,u)$ since at least one neighbor of u is not within its antenna beam of angle ϕ. Add $\{v,w\}$ to G to form a cycle C_u; see Figure 5. We "orient" C_u in $\overrightarrow{G}$ along any one direction (all the arcs in C_u in the opposite direction are removed). This process does not break the strong connectivity of $\overrightarrow{G}$. Finally, if $|C_u| > 3$, we remove from S every vertex that is in C_u. However, if $|C_u| = 3$, we only remove u from S. We repeat this process until S is empty.

Let $\mathcal{C}$ be the set of cycles that are formed with the addition to T of the new edges. We will prove that after adding each new cycle $C_u \in \mathcal{C}$ of hop-length greater than 3 the angle

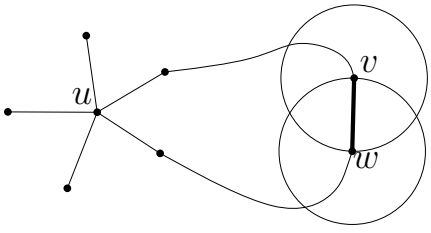


Fig. 5. The shortest edge $\{v,w\}$ connecting two components of $G \setminus \{u\}$ forms a cycle

that any two points form with a common neighbor in G is at least $\pi/3$. Indeed, since C_u is formed with the smallest edge $\{v,w\}$ connecting two components, $D(v;d(v,w)) \cap D(w;d(v,w))$ is empty. Therefore, the min angle that $\{v,w\}$ forms with the neighbors of v and w is at least $\pi/3$. Furthermore, since points are in general position the degree of the vertices in C_u is at most five. Therefore, the out-degree in $\overrightarrow{G}$ of every vertex in C_u is at most four. (At least one in-going edge in $\overrightarrow{G}$.)

Now we consider a cycle C_u of hop-length three. Let $\{v,w\}$ be the shortest edge connecting two components of $G \setminus \{u\}$. Observe that both $D(u;d(u,v)) \cap D(v;d(u,v))$ and $D(u;d(u,w)) \cap D(w;d(u,w))$ are empty. However, $D(v;d(v,w)) \cap D(w;d(v,w))$ contains u. Therefore, the min angle that each edge incident to u,v and w forms with an edge of the triangle uvw is at least $\pi/3$. Thus, we reduce the out-degree of u to at most four since the points are in general position. Moreover, the out-degree of v and w remains the same. However, since v and w are not removed from S when C_u is created, they are included in distinct cycles provided that they are in S.

As in the proof of Theorem 2, we can show that the construction of $\overrightarrow{G}$ can be implemented in $O(n^2)$ time. Indeed, the edges to be added are always edges of either the Delaunay Triangulation on P or the closest neighbors of each vertex. Thus, the addition of each edge takes time $O(n)$. The theorem follows since $|S| = O(n)$ and the orientation of the antennae takes time $O(1)$.

For the next theorem we use the main result of [6][Theorem 1]. For convenience we state this theorem without proof.

Theorem 6 (k-Antennae Orientation [6].). *Consider a set S of n sensors in the plane and suppose each sensor has k, $1 \leq k \leq 5$, directional antennae with any angle $\phi \geq 0$. Then the antennae can be oriented at each sensor so that the resulting spanning graph is strongly connected and the range of each antenna is at most $2\sin\left(\frac{\pi}{k+1}\right)$ times the optimal. Moreover, given a MST on the set of points the spanner can be constructed with additional $O(n)$ overhead.*

The following theorem shows that for any angle $\phi \geq \pi/2$ we can always construct a strongly connected transmission network with longest edge bounded by $\sqrt{3}$ times the longest edge of the MST.

Theorem 7. *There is an algorithm which for any set of n points in the plane, orients double antennae of beamwidth $\frac{\pi}{2} \leq \phi \leq \frac{2\pi}{3}$ using range bounded by $\sqrt{3}$ times the optimal range so that the resulting graph is strongly connected.*

Proof. Let $\overrightarrow{G}$ be the strongly connected digraph with out-degree 2 (i.e., using $k = 2$ antennae) and range bounded by $\sqrt{3}$ obtained from Theorem 6. Since the out-degree of $\overrightarrow{G}$ is bounded by two, from Lemma 4 a double antenna with angle at most $\pi/2$ covers the two out-going edges. This completes the proof of the theorem.

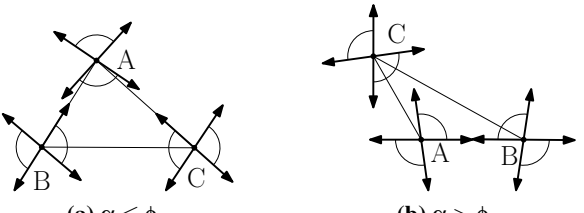
5 Stretch Factor

In this section, we consider the stretch factor problem, that is, finding an orientation of double antennae of minimum possible range that induces a c-directional spanner, for some constant c. That is, given a set P of points in the plane such that $UDG(P,1)$ is connected, we wish to replace omnidirectional antennae of range 1 with double antennae of angle ϕ and range r such that for any edge in $UDG(P,1)$, there is a path in the resulting strongly connected digraph of length at most c for some constant c. The basic idea is to partition the set of points into triples such that in each triple, there is at most one pair of vertices that is not connected in the UDG. For each triple $\{A,B,C\}$ we need to determine the antenna range r required so that there is an orientation of three directional antennae placed at A,B,C, respectively, so that every point within distance two of at least one of the points A,B,C is also within "directional antenna range" of radius r from at least one of these three points.

First we prove a basic lemma concerning double antenna orientation of three points A,B,C in the plane. The antenna orientation will depend on the largest angle, say $\alpha = \angle(BAC)$, that the three points form.

Lemma 5. *Consider three points A,B,C in the plane forming a triangle. Three identical double antennae of beamwidth $\phi \geq \pi/2$ can be oriented so as to cover the whole plane.*

Proof. We consider two cases depending on the size of α.



(a) $\alpha \leq \phi$. **(b)** $\alpha > \phi$.

Fig. 6. Orientation of three double antennae

Case $\alpha \leq \phi$. Without loss of generality asume that BC is horizontal and A is above BC. Orient the antennae as depicted in Figure 6a so that the antenna covers the triangle and the wedge of the antennae at B and C are on BC and CA respectively. Observe that the antennae cover the "whole plane" since each angle of the triangle is always covered.
Case $\alpha > \phi$. Without loss of generality asume that AB is the second smallest edge in the triangle, AB is horizontal and C is above AB. Orient the antennae as depicted in

Figure 6b so that the one antenna wedge of C is vertical and the wedge of the antennae at A and B are on AB. To prove that the orientation covers "the whole plane", observe that the antennae at A and B only leave a black (i.e., uncovered) corridor in the lower half-plane determined by AB. However, the antenna at C covers the black corridor. This completes the proof of the lemma.

We now consider double antennae of finite range. The following results hold for double antennae of range r, but can also be shown to hold for the weaker model of $(\phi, r, 2)$-double antennae. The proof of the following lemma is omitted due to space constraints.

Lemma 6. *Let A, B, C be three points such that $d(A,B) \leq 1$ and $d(A,C) \leq 1$. Assume $\frac{\pi}{2} \leq \phi \leq \pi$. We can orient three $(\phi, r, 2)$-double antennae (Yagi-like antennae) of beam width ϕ at A, B, C so that every point at distance at most two from one of these points is covered by one of the three antennae, where $r \leq 4\sin\left(\frac{\pi}{4} + \frac{\phi}{2}\right)$.*

Theorem 8. *Given $\frac{\pi}{2} \leq \phi < \pi$, there is an algorithm which for any connected $UDG(P)$ on a set P of points in the plane, orients $\left(\phi, 4\sin\left(\frac{\pi}{4} + \frac{\phi}{2}\right), 2\right)$-double antennae so that the resulting graph has stretch factor four. Furthermore, it can be done in linear time.*

Proof. Let $\mathcal{T}$ be any partition of the UDG with the maximal number of triples such that every triangle has two edges of length at most one. It is easy to see that such a partition can be constructed in linear time. For each triangle T in $\mathcal{T}$, we orient the antennae at T as shown in Lemma 5 with range $4\sin\left(\frac{\pi}{4} + \frac{\phi}{2}\right)$ and the antenna of each remaining sensor toward its nearest triangle. Observe that the closest triangle is at distance at most two. Let $\overrightarrow{G}$ be the strongly connected network induced by the antennae. We will prove that for each edge $\{u, v\} \in UDG(P)$, there is a directed path P from u to v and a directed path P' from v to u of hop-length no more than 4 hops. Let T and T' be in two different triangles in the partition $\mathcal{T}$.

- $u, v \in T$. Then $|P| \leq 2$ and $|P'| \leq 2$.
- $u \in T$ and $v \in T'$. Since $d(u,v) \leq 1$, v is in the coverage area of T. Therefore, u can reach v in at most three hops and $|P| \leq 3$. A similar argument shows that $|P'| \leq 3$.
- At least one of u or v is not in any triangle of $\mathcal{T}$. Assume without loss of generality that u is not in a triangle. Observe that there exists a triangle T at distance at most two from u. Otherwise, $\mathcal{T}$ is not maximal. Therefore, u can reach v through T in at most four hops, i.e., $|P| \leq 4$. Similarly, we can prove that $|P'| \leq 4$.

This completes the proof of the theorem.

6 Conclusion

In this paper we considered algorithms for orienting antennae with Dipole-like and Yagi-like antenna propagation patterns so as to attain optimal connectivity and stretch factor of the resulting directed network. It would be interesting to improve the bound in our connectivity results for the range $[\pi/3, \pi/2]$, and to prove better bounds for the stretch factor problem either in terms of range or in terms of stretch factor.

References

1. Bose, P., Carmi, P., Damian, M., Flatland, R., Katz, M.J., Maheshwari, A.: Switching to Directional Antennas with Constant Increase in Radius and Hop Distance. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 134–146. Springer, Heidelberg (2011)
2. Caragiannis, I., Kaklamanis, C., Kranakis, E., Krizanc, D., Wiese, A.: Communication in wireless networks with directional antennas. In: Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures, pp. 344–351. ACM (2008)
3. Cormen, T.H.: Introduction to Algorithms, 2nd edn. The MIT Press (2007)
4. Damian, M., Flatland, R.: Spanning properties of graphs induced by directional antennas. In: Electronic Proc. 20th Fall Workshop on Computational Geometry. Stony Brook University, Stony Brook (2010)
5. De Berg, M., Cheong, O., Van Kreveld, M.: Computational geometry: algorithms and applications. Springer-Verlag New York Inc. (2008)
6. Dobrev, S., Kranakis, E., Krizanc, D., Opatrny, J., Ponce, O.M., Stacho, L.: Strong Connectivity in Sensor Networks with Given Number of Directional Antennae of Bounded Angle. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 72–86. Springer, Heidelberg (2010)
7. Fleischner, H.: The square of every two-connected graph is hamiltonian. Journal of Combinatorial Theory, Series B 16(1), 29–34 (1974)
8. Gupta, H., Kumar, U., Das, S.R.: A topology control approach to using directional antennas in wireless mesh networks. IEEE International Conference on Communications 9(06), 4083–4088 (2006)
9. Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Transactions on Information Theory 46(2), 388–404 (2000)
10. Hu, L., Evans, D.: Using directional antennas to prevent wormhole attacks. In: Network and Distributed System Security Symposium, NDSS (2004)
11. Kranakis, E., Krizanc, D., Morales, O.: Maintaining connectivity in sensor networks using directional antennae. In: Nikoletseas, S., Rolim, J. (eds.) Theoretical Aspects of Distributed Computing in Sensor Networks, pp. 59–84 (2010)
12. Kranakis, E., Krizanc, D., Williams, E.: Directional Versus Omnidirectional Antennas for Energy Consumption and *k*-Connectivity of Networks of Sensors. In: Higashino, T. (ed.) OPODIS 2004. LNCS, vol. 3544, pp. 357–368. Springer, Heidelberg (2005)
13. Parker, R.G., Rardin, R.L.: Guaranteed performance heuristics for the bottleneck travelling salesman problem. Operations Research Letters 2(6), 269–272 (1984)
14. Schiller, J.H.: Mobile Communications. Addison Wesley (2003)
15. Yi, S., Pei, Y., Kalyanaraman, S., Azimi-Sadjadi, B.: How is the capacity of ad hoc networks improved with directional antennas? Wireless Networks 13(5), 635–648 (2007)

Distributed Multiple-Message Broadcast in Wireless Ad-Hoc Networks under the SINR Model

Dongxiao Yu[1], Qiang-Sheng Hua[2], Yuexuan Wang[2], Haisheng Tan[2], and Francis C.M. Lau[1]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong, P.R. China
[2] Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, 100084, P.R. China

Abstract. In a multiple-message broadcast, an arbitrary number of messages originate at arbitrary nodes in the network at arbitrary times. The problem is to disseminate all these messages to the whole network. This paper gives the first randomized distributed multiple-message broadcast algorithm with worst-case performance guarantee in wireless ad-hoc networks employing the SINR interference model which takes interferences from all the nodes in the network into account. The network model used in this paper also considers the harsh characteristics of wireless ad-hoc networks: there is no prior structure, and nodes cannot perform collision detection and have little knowledge of the network topology. Under all these restrictions, our proposed randomized distributed multiple-message broadcast protocol can deliver any message m to all nodes in the network in $O(D + k + \log^2 n)$ timeslots with high probability, where D is the network diameter, k is the number of messages whose broadcasts overlap with m, and n is the number of nodes in the network. We also study the lower bound for randomized distributed multiple-message broadcast protocols. In particular, we prove that any uniform randomized algorithm needs $\Omega(D + k + \frac{\log^2 n}{\log\log\log n})$ timeslots to deliver k messages initially stored at k nodes to all nodes in the network.

1 Introduction

In wireless networks, how to achieve efficient communications is one of the most extensively studied problems. The main challenge is to deal with interferences. Hence, the modeling of wireless interferences will play a fundamental role in designing efficient network protocols. Previous work mostly adopted the graph based or the protocol interference model. In the graph based interference model, it is assumed that only nodes within d (a small constant) hops from a receiver can interfere with the transmission. The protocol model assumes that a transmission can be successful if and only if there is only one transmitter within a certain range centered at the receiver. A shortcoming of these two types of models is that they treat interference as a localized phenomenon, which is not likely

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 111–122, 2012.

the case in practice. In real wireless networks, the interference is cumulative, being contributed to by all simultaneously transmitting nodes. Because of the lack of the ability to capture the cumulative property of interference, the protocols designed under the graph based or protocol model display a dramatically different performance from the expectation in practice. In this paper, we adopt the SINR model (also known as the physical interference model since it reflects the physical reality more accurately), which defines a global interference function and takes into account the cumulative property of interference. Besides interference, some other important aspects in wireless ad-hoc networks should also be considered when modeling the network. For instance, when the network begins operation, no built-in infrastructure or MAC layer is available to the nodes to facilitate communication between neighboring nodes, and the nodes are clueless about the network topology. Furthermore, the nodes may not be able to perform any type of collision detection because they may be just tiny sensors with limited capabilities and energy [15].

In the multiple-message broadcast problem, an arbitrary number of messages arrive at arbitrary nodes from the environment at arbitrary times. The problem is to deliver all these messages to all the nodes. A multiple-message broadcast protocol can be used as a building block for many applications, e.g., update of routing tables, topology learning of the underlying network, and aggregating functions in sensor networks.

Different from most previous work, in this work, we do not assume that all messages are initially stored at their nodes. In addition, we adopt the realistic global SINR interference model and assume the imposition of such rigorous restrictions as no prior structure, no collision detection and nodes have little knowledge about the network topology; all these add to the challenge of designing an efficient distributed protocol. Under all these rigorous but practical restrictions, we present a randomized distributed multiple-message broadcast algorithm for wireless ad-hoc networks, and show that, with high probability, any message m can be broadcast to all nodes in $O(D+k+\log^2 n)$ timeslots after its arrival, where D is the diameter of the communication graph defined by the maximum transmission range R_M (refer to Section 3), k is the number of messages whose broadcast overlap m (refer to Section 3) and n is the number of nodes in the network.[1] To the best of our knowledge, this work is the first one that studies time efficient distributed multiple-message broadcast algorithms in wireless ad-hoc networks under the SINR model. Our result significantly surpasses the best known results of $\max\{O(k\log n\log\Delta+(D+n/\log n)\log n\log\Delta), O((k\Delta\log n+D)\log\Delta)\}$ [1,8] under the graph based interference model, and breaks the expected $\Omega(k+D\log(n/D))$ lower bound [4,10] for randomized solutions under the graph-based radio network model. Note that the previous results are obtained

[1] We define the running time of a multiple-message broadcast algorithm as the maximum number of timeslots to disseminate a message to the whole network. If all messages are initially stored at their nodes, our defined performance measurement is equivalent to that in previous work [1], which is the number of timeslots needed to broadcast all messages to all nodes.

with knowledge of some network parameters, e.g., Δ and D; In contrast, our algorithm does not assume any prior information concerning such parameters. The trivial $\Omega(D+k)$ lower bound indicates that our algorithm is asymptotically optimal for networks with diameter $D \in \Omega(\log^2 n)$.

Besides the proposed algorithm, we also study the lower bound of the time needed by randomized distributed algorithms to accomplish multiple-message broadcast. Specifically, we show that if all the nodes use the same transmission power, any uniform randomized algorithm in which all awaken nodes transmit a message with the same probability (independent of the communication history) in every timeslot [3] needs $\Omega(D+k+\frac{\log^2 n}{\log\log\log n})$ timeslots to accomplish multiple-message broadcast even under the assumption that all messages are initially stored at their nodes.

2 Related Work

Although the SINR model (or the physical interference model) poses great challenges for designing efficient distributed algorithms due to its global interference, there have been some attempts in recent years. In [14], with the assumption that all nodes can perform physical carrier sensing, an $O(\log n)$ time randomized distributed algorithm for computing a constant approximate dominating set was presented. The local broadcasting problem was first considered in [5]. In this paper, based on whether each node knows the number of nodes in its proximity region or not, the authors gave two randomized distributed algorithms with approximation ratios $O(\log n)$ and $O(\log^3 n)$, respectively. The latter result was improved by some recent papers [19,16], the latter of which achieves an approximation ratio of $O(\log n)$. By assuming that nodes can perform physical carrier sensing, the authors of [19] also gave two distributed deterministic local broadcasting algorithms both having an approximation ratio of $O(\log n)$ for asynchronous wake-up and synchronous wake-up scenarios. The distributed $(\Delta+1)$-coloring (Δ is the maximum network degree) was studied in [18] and an $O(\Delta \log n + \log^2 n)$ time randomized distributed algorithm was given. There are also recent papers on finding efficient distributed algorithms for the minimum latency aggregation scheduling problem [12,13] and the wireless scheduling problem [11,6].

The multiple-message broadcast problem is also called the the Many-to-All communication problem [4]. All previous work assumes the standard graph-based radio network model. In this model, there is a link existing between any pair of nodes that can communicate with each other. A transmission is successful iff there is only one neighbor transmitting a message to the receiver. Additionally, except [8,9], all work assumes that all messages are stored at their nodes at the beginning of the algorithm. The authors of [1] first initiated the study of this problem. They designed a randomized algorithm accomplishing multiple-message broadcast in $O(k \log n \log \Delta + (D + n/\log n)\log n \log \Delta)$ rounds in expectation. Assuming nodes receive messages at arbitrary times from the environment, the authors of [8] proved that their modular approach can

broadcast a message to all nodes in $O((k\Delta \log n + D)\log \Delta)$ rounds with high probability when there are at most k concurrent messages. How to use network coding techniques to accelerate the multiple-message broadcast has been studied in [7], in which the proposed randomized algorithm achieves a time complexity of $O(k \log \Delta + (D + \log n)\log \Delta \log n)$. All the above work assume that nodes know some or all network parameters, e.g., Δ and D. The best known lower bound for randomized solutions under the graph-based radio network model is $\Omega(k + D\log(n/D))$ in expectation [4,10]. In the paper [9], by introducing an abstract MAC layer providing reliable local broadcast communication, the authors gave a multiple-message broadcast protocol for regional networks and showed that the protocol can broadcast a message to all nodes in $O((D + k)F_{prog} + (k - 1)F_{ack})$ rounds, where F_{prog} and F_{ack} are progress and acknowledgement bounds respectively.

3 Network Model and Problem Definitions

We assume there are n processors; they are the nodes of the network. During the protocol execution, the time is divided into slots. Processors have synchronized clocks and they have access to a global clock. We also assume that all processors wake up at the beginning of the protocol. We do not assume any placement distribution for nodes, i.e., nodes are arbitrarily placed on the plane. At the beginning, the network is completely unstructured. Nodes have very little information about the network topology. They have no knowledge about their neighbors, even the number of nodes in their proximity range. Only a polynomial estimate n of the number of nodes in the network is given to the nodes. Nodes have no collision detection mechanism. In other words, nodes can not distinguish between the occurrence of a collision and the case that there are no transmissions. We also assume that each node has a unique ID. The IDs need not be in the interval $[1, n]$, which are only used for a receiver to identify its sender. Furthermore, we assume that there is only one channel available and nodes operate in half-duplex mode, i.e., in a timeslot, a node can only carry out either one of the two operations: receive and transmit.

We adopt the SINR interference model. In this model, a message sent by node u to node v can be correctly received at v iff

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

where P_u (P_w) is the transmission power of node u (w); α is the path-loss exponent whose value is normally between 2 and 6; β is a hardware determined threshold which is greater than 1; N is the ambient noise; $d(u, v)$ denotes the Euclidean distance between u, v and $\sum_{w \in V \setminus \{u,v\}} \frac{P_w}{d(w,v)^\alpha}$ is the accumulated interference experienced by the receiver v caused by all other simultaneously transmitting nodes in the network.

Given transmission power P for v, the transmission range R_T of a node v is defined as the maximum distance at which a node u can receive a clear transmission

from v ($SINR \geq \beta$) when there are no other simultaneous transmissions in the network. According to (1), $R_T \leq (\frac{P}{\beta \cdot N})^{1/\alpha}$. We further define $R_T = (P/cN\beta)^{1/\alpha}$, where $c > 1$ is a constant determined by the environment. Based on the transmission ranges of nodes, we define a communication graph $G = (V, E)$, where V is the set of nodes in the network, and a link $(u, v) \in E$ if and only if $d(u, v)$ is not larger than the transmission range of u. Furthermore, if all nodes have the same transmission range R_T, the obtained communication graph is denoted as G_{R_T}. Obviously, in this case, G_{R_T} can be seen as an undirected graph. We say a network is connected in terms of R if the communication graph G_R is connected. Let P_M and R_M be the maximum transmission power and the corresponding maximum transmission range of nodes respectively. Denote D as the diameter of the communication graph G_{R_M}.

Given a distance d, we say two nodes are independent if the distance between them is larger than d. An independent set I in terms of d is defined as a set of nodes such that any pair of nodes in I are independent. An independent set I is maximal in terms of d if for any node v in the network, either $v \in I$, or there is a node in I that is within distance d from v. A dominating set S in terms of d is defined as that for any node v, either $v \in S$, or there is a node in S that is within distance d from v. Denote G_d^S as the subgraph of G_d induced by S. A dominating set S is said to be connected in terms of d if G_d^S is connected. Note that a maximal independent set is a dominating set, but not a connected dominating set.

For a message m, denote $arrive(m)$ as the event that the message m arrives at the network, i.e., m is received by some node v. Denote $clear(m)$ as the event that the network has completed the broadcast of message m, i.e., all nodes in the network have received m. Then $K(m)$ is used to denote the set of messages whose processing overlaps with the interval between $arrive(m)$ and $clear(m)$. In other words, $K(m)$ is the set of messages m' such that an $arrive(m')$ event precedes the $clear(m)$ event and the $clear(m')$ event follows the $arrive(m)$ event. Let $k = |K(m)|$.

4 Algorithm

4.1 Algorithm Description

The proposed multiple-message broadcast algorithm adopts a clustering strategy, which encompasses four processes: leader election, leader coloring, local information collection and broadcast. The whole algorithm is divided into three stages. The first stage works as a pre-stage in which a CDS (Connected Dominating Set) in terms of distance R is computed, where $R = \min\{\frac{1}{2}, (\frac{48c\beta(2^{\alpha-1}+\frac{\alpha-1}{\alpha-2})}{c-1})^{-\frac{1}{\alpha}}\} \cdot R_M$. Nodes in the computed connected dominating set are called leaders. Other nodes are called non-leaders, each of which chooses the first leader that successfully transmits a dominating message to it as its leader. A cluster is composed of a leader and its dominated non-leaders. The parameter R is chosen to guarantee that if there is only one sender in each disk with radius R_M, each sender can successfully transmit a message to all nodes within distance R. The second stage is used to get a TDMA-like scheduling scheme for nodes to perform the local information collection process and the broadcast process in the next stage

by performing a coloring. In the third stage, each leader first collects messages that are received from the environment by its dominated non-leaders. Then the messages are disseminated to the whole network through the backbone network composed by the leaders. During the execution of the protocol, each leader v is assigned a queuing set Q_v to store received messages. Next we describe the algorithm in more details.

Stage 1 Leader Election: This stage is to compute a connected dominating set in terms of R, the nodes of which form a backbone network for performing the broadcast in the Stage 3. As shown in [2], for a graph G, if we find connectors such that any pair of MIS (Maximal Independent Set) nodes within three hops are connected by these connectors, the MIS nodes and the connectors constitute a CDS. In this stage, the nodes first execute the MIS algorithm in [18] to compute an MIS in terms of $R/3$. Any two nodes within three hops in the computed MIS have distance at most R, so they are connected in terms of R. This stage takes $O(\log^2 n)$ timeslots. By the end of this stage, a CDS in terms of R is correctly computed with high probability[2]. Furthermore, we will show that the computed CDS satisfies the property that in any disk with radius R_M, there are only a constant number of leaders. Denote χ as a constant upper bound for the number of leaders in a disk with radius R_M. The value of χ will be given later.

Stage 2 Leader Coloring: In this stage, we want to find a coloring for leaders in the computed connected dominating set, such that for any two leaders, if the distance between them are not larger than R_M, they get different colors. Since for any leader, there are at most $\chi - 1$ other leaders within distance R_M, χ colors are enough to color all the leaders. We use a greedy coloring algorithm to accomplish the coloring process. The MIS algorithm in [18] is iteratively executed to get an MIS in terms of R_M from leaders that have not been colored. In the i-th execution, an MIS in terms of R_M is obtained from the remaining uncolored leaders and its nodes are assigned the color i. Finally, each leader gets a color from $\{1, 2, \ldots, \chi\}$. After the coloring is computed, there are χ timeslots for the leaders to inform their dominated non-leaders of their colors. Since each execution of the MIS algorithm needs $O(\log^2 n)$ timeslots, this stage takes at most $O(\chi \log^2 n) \in O(\log^2 n)$ timeslots.

Stage 3 Local Information Collection and Broadcast: This stage is for leaders to collect the messages that arrive at their dominated non-leaders and then disseminate the received messages to the whole network. The stage is divided into iterative substages, each of which consists of χ phases. In each substage, the i-phase is for leaders with color i to accomplish information collection and broadcast. Each phase has three timeslots. This TDMA-like scheduling makes sure that in each phase, any two leaders that are active have distance larger than R_M, so that each leader can successfully transmit an acknowledgement message to its dominated non-leaders and send the received message to all nodes within distance R. During the execution of this stage, all nodes use the same transmission power

[2] We assume that the network is connected in terms of $R/3$, i.e., the communication graph $G_{R/3}$ is connected.

$P_B = cN\beta R^\alpha$, which leads to the same transmission range R (refer to Section 3). Next we describe the detailed operations in the i-th phase of a substage.

The i-th Phase: In this phase, leaders with color i and non-leaders in their clusters execute a three-timeslot scheme as described in Algorithm 1. The first two timeslots are used for local information collection and the third is for the leaders to broadcast the received messages. In particular, in the first timeslot, each non-leader that has received a message from the environment endeavors to transmit the message to its leader with a specified transmission probability. Here the transmission probability is initially set as a small value $\frac{\lambda}{n}$, where λ is a constant to be given later. After Stage 3 has started, every non-leader u that is performing the local information collection process updates its transmission probability at every $9\chi\lambda^{-1}4^{2(\lambda+1)}\log n$-th timeslot as shown in Algorithm 1. The updating principle is set to guarantee that on one hand, nodes can increase the transmission probability, by which they can finally get a large enough transmission probability ensuring a successful transmission; on the other hand, the sum of transmission probabilities of nodes in any local region will not exceed a constant which is the base of obtaining a sufficient condition for successful transmissions. Furthermore, while a non-leader transmits the message, it also adds its ID and its leader's ID into the transmitted packet such that its leader can distinguish whether the received message is for it or not. In the second timeslot, if a leader v receives a message from one of its dominated non-leaders u, it stores the received message into Q_v and transmits an $Ack_v(u)$ message to inform u that it has received the message. A leader v also adds the messages received from other leaders into Q_v. After receiving the $Ack_v(u)$ message, u will stop transmitting and quit the local information collection process. In the third timeslot, for each leader v with color i, if Q_v is not empty, it transmits the first message in Q_v to all nodes within distance R and deletes it from Q_v.

In order to ensure that the above described multiple-message broadcast algorithm is correct with high probability, we set the parameters as follows: $\lambda = \frac{(1-1/c)}{192\beta} \cdot (2^{\alpha-1} + \frac{\alpha-1}{\alpha-2})^{-1}$, and $\chi = (\frac{6R_M}{R} + 1)^2$.

4.2 Analysis

In this section, we prove that with probability $1 - O(\frac{1}{n})$, for any message m, the proposed multiple-message broadcast algorithm can disseminate m to the whole network after $arrive(m)$ occurs for at most $O(D + k + \log^2 n)$ timeslots. Before starting the analysis, we first define some notations. We use T_v and I_v to denote the disks of radii R and R_M centered at node v, respectively. The notation E_v^d denotes the disk of radius d centered at v. Without confusion, we also use these notations to denote the nodes in the corresponding disks.

The following lemma is proved in [18], which states the correctness and efficiency of the MIS algorithm.

Lemma 1. *After executing the MIS algorithm for $O(\log^2 n)$ timeslots, a maximal independent set can be correctly computed with probability at least $1-O(n^{-1})$.*

In the following, we assume that the MIS is correctly computed, and the error probability will be summed up in the proof of the main theorem (Theorem 1).

Using a standard area argument, we can give a constant bound on the number of leaders in any disk I_i as shown in Lemma 2 which follows. For the detailed proof of the lemma, please refer to the full version [17].

Lemma 2. *In a disk I_i with radius R_M, the number of leaders is at most χ.*

Algorithm 1. 3-Timeslot Scheme

Initially, $p_u = \frac{\lambda}{2n}$; $Q_v = \emptyset$;
3-timeslot scheme for a leader v
1: listen
2: **if** v received a message from a non-leader u in its cluster **then** transmit $Ack_v(u)$ **end if**
3: **if** Q_v is not empty **then** transmit the first message in Q_v and delete the message from Q_v **end if**
Message Received
4: **if** v received a message that has not been received **then** add the message into Q_v **end if**
3-timeslot scheme for a non-leader u
5: **if** u has a message received from the environment **then** transmit the message with probability p_u **end if**
6: listen
7: **if** u received $Ack_v(u)$ **then** stop transmitting and quit the local information collection process **end if**
Update p_u
8: **while** $t = i \cdot 9\chi\lambda^{-1}4^{2(\lambda+1)} \log n$ for some integer $i > 0$
9: **if** u has taken part in the local information collection process and received less than $12 \log n$ *Ack* messages from its leader in the past $9\chi\lambda^{-1}4^{2(\lambda+1)} \log n$ timeslots **then** $p_u = 2p_u$
10: **else** $p_u = p_u/2$
11: **end if**
12: **end while**

Lemma 3. *In Stage 2, the coloring process only needs to execute the MIS algorithm for at most χ times, and each leader can get a color from $\{1, 2, \ldots, \chi\}$.*

Proof. By Lemma 2, for each leader v, there are at most $\chi - 1$ leaders within distance R_M. From the coloring process, we know that after executing the MIS algorithm once, for each leader v, either v joins the computed MIS and gets a color, or a leader in I_v joins the MIS and gets a color. Thus after executing the MIS algorithm for at most $\chi - 1$ times, either v has chosen its color, or all leaders in I_v have been colored. For the second case, v will join the MIS in the next execution of the MIS algorithm and will get a color from $\{1, 2, \ldots, \chi\}$. □

By Lemma 1 and Lemma 3, we have the following corollary which states the time needed for executing Stage 1 and Stage 2.

Corollary 1. *With probability $1 - O(n^{-1})$, the coloring computed in Stage 2 is correct, and the number of timeslots needed for Stage 1 and Stage 2 is $O(\log^2 n)$.*

Next we analyze the correctness of Stage 3 and the number of timeslots needed for implementing Stage 3. We first state in the following Lemma 4 that making use of the TDMA-like scheduling as shown in Stage 3 can guarantee successful local broadcast within distance R for a leader v. Due to the space limitation, the proof is given only in the full version [17].

Lemma 4. *In Stage 3, during executing the three-timeslot scheme, a leader v can successfully transmit a message to all its neighbors within distance R in the second and third timeslots.*

Before analyzing the timeslots needed for Stage 3, we first present the following property which gives a constant bound on the sum of transmission probabilities of non-leaders in any disk T_v centered at some leader v. The idea of the proof is to show in any disk T_v, when the sum of transmission probabilities of non-leaders is about to break the declared bound, with high probability, every non-leader in T_v must have received at least $12 \log n$ *Ack* messages from the leader in the past $9\chi\lambda^{-1}4^{2(\lambda+1)} \log n$ timeslots. Then by the algorithm, these nodes will halve their transmission probabilities, which gurantees that the declared bound will not be broken during the execution of the algorithm with high probability. For the detailed proof, please refer to [17].

Property 1. In any timeslot during the execution of Stage 3, for any leader v, the sum of transmission probabilities of non-leaders in T_v is at most 2λ.

Based on the above property, we give an upper bound on the number of timeslots needed for the local information collection process in the following Lemma 5. Denote Δ_k^v as the number of messages in $K(m)$ that arrive at nodes within distance R from v. Let $\Delta_k = \max\{\Delta_k^v\}$ for all nodes v. Clearly, $\Delta_k \leq k$.

Lemma 5. *Assume that Property 1 holds. For a non-leader u, it can transmit its message to the leader after starting transmission for $O(\Delta_k + \log^2 n)$ timeslots with probability $1 - O(n^{-2})$.*

Proof. We first give a sufficient condition for a successful transmission from a non-leader u to its leader v in the following claim whose proof can be found in the full version [17].

Claim. If a non-leader u is the only transmitting node in T_v, v can successfully receive the message sent by u with probability at least $\frac{1}{2}$.

Next we bound the number of timeslots for u to successfully transmit its message to the leader. After Stage 3 has started, once a non-leader u receives a message from the environment, it transmits its message to the leader v in the corresponding phases by executing the three-timeslot scheme. As shown in Algorithm 1, after u starts executing the three-timeslot scheme, in every $9\chi\lambda^{-1}4^{2(\lambda+1)} \log n$ timeslots, either u receives at least $12 \log n$ *Ack* messages from v and halves its transmission probability, or u doubles its transmission probability. Thus, after $6\chi\lambda^{-1}4^{2\lambda+1}\Delta_k + 9\chi\lambda^{-1}4^{2(\lambda+1)} \log^2 n$ timeslots, if u does not receive an $Ack_v(u)$ message, it must have a constant transmission probability $\lambda/2$, since u halves its transmission probability for at most $\frac{\Delta_k}{12 \log n}$ times, and therefore it needs $\frac{\Delta_k}{12 \log n} \cdot 9\chi\lambda^{-1}4^{2(\lambda+1)} \log n = 3\chi\lambda^{-1}4^{2\lambda+1}\Delta_k$ timeslots to increase the transmission probability to the initial value. Then based on the sufficient condition for

a successful transmission in the above claim, we will show that the probability P_{no} that u can not send its message to v in the subsequent $6\chi\lambda^{-1}4^{2\lambda+1}\log n$ timeslots is at most $O(n^{-2})$. Let P_{only} denote the probability that u is the only transmitting node in T_v, and we have

$$\begin{aligned} P_{only} &\geq p_u \prod_{w\in T_v\setminus\{u\}}(1-p_w) \geq p_u \cdot \left(\frac{1}{4}\right)^{\sum_{w\in T_v\setminus\{u\}} p_w} \\ &\geq \frac{\lambda}{2}\left(\frac{1}{4}\right)^{\sum_{w\in T_v} p_w} \geq \frac{\lambda}{2}\left(\frac{1}{4}\right)^{2\lambda}. \end{aligned} \tag{2}$$

The last inequality is derived by Property 1. So the probability that u can not send its message to v in the subsequent $2\lambda^{-1}4^{2\lambda+1}\log n$ transmissions is at most $P_{no} \leq (1-\frac{1}{2}\cdot\frac{\lambda}{2}\left(\frac{1}{4}\right)^{2\lambda})^{2\lambda^{-1}4^{2\lambda+1}\log n} \leq e^{-\frac{\lambda}{4}\left(\frac{1}{4}\right)^{2\lambda}\cdot 2\lambda^{-1}4^{2\lambda+1}\log n} \leq n^{-2}$.

Therefore, after starting transmission for $6\chi\lambda^{-1}4^{2\lambda+1}\Delta_k + 9\chi\lambda^{-1}4^{2(\lambda+1)}\log^2 n + 6\chi\lambda^{-1}4^{2\lambda+1}\log n$ timeslots, with probability $1-n^{-2}$, u must have successfully transmitted its message to v, since each non-leader transmits once in every 3χ timeslots. □

The following Lemma 6 is given in [9], which analyzes the pipelining effect of the multiple-message broadcast process. Let F_{prog} denote the maximum number of timeslots needed for a successful transmission. For a graph G, define $d_G(u,v)$ as the number of edges in the shortest path from u to v in G.

Lemma 6. *Assume that in timeslot t_0, a node u receives a new message m. Let v be a node at distance $d = d_G(u,v)$ from v. For integers $l \geq 1$, we define $t_{d,l} = t_0 + (d+2l-2)F_{prog}$. Then for all integers $l \geq 1$, at least one of the following two statements is true:*

(i) v received the message m by the time $t_{d,l}$;

(ii) there exists a set $M \subseteq K(m)$, $|M| = \min\{l,k\}$, such that for every $m' \in M$, v has received m' by the timeslot $t_{d,l}$.

Theorem 1. *With probability $1-O(n^{-1})$, any message m can be broadcast to all nodes in the network after the event arrive(m) occurs for $O(D+k+\log^2 n)$ timeslots.*

Proof. By Corollary 1, Stage 1 and Stage 2 need $O(\log^2 n)$ timeslots. So if m arrives at the network before Stage 3 starts, it waits for $O(\log^2 n)$ timeslots before Stages 1 and 2 complete. Next we analyze the timeslots needed for m to be broadcast to the whole network in Stage 3.

As proved in Lemma 5, in Stage 3, if m arrives at a non-leader u, with probability $1-O(n^{-2})$, u can send m to its leader in $O(\Delta_k+\log^2 n)$ timeslots. And after that, m will be broadcast by each leader that received m to all its neighbors within distance R in the corresponding phases. If m arrives at a leader, the broadcast process starts after its arrival. By Algorithm 1, each leader broadcasts one message for one timeslot in every χ phases. And by Lemma 4, a leader can successfully transmit the message to all its neighbors within distance R. So it only takes a constant number of timeslots for m to be propagated for one hop in

the communication graph G_R. Note that R is constant fraction of R_M, the diameter of G_R is $O(D)$. Then by Lemma 6, after $O(D+2k-2)$ timeslots, either all nodes have received m, or all nodes have received k messages. By the definition of $K(m)$ and $k = |K(m)|$, m must have been received by all nodes. Combining all these, with probability $1-O(n^{-2})$, m can be broadcast to the whole network after arriving at the network for $O(D+k+\log^2 n)$ timeslots. Since k is at most $O(n)$, this is true for any message with probability $1-O(n^{-1})$.

Note that all the above discussions are based on Property 1 and the assumption that the connected dominating set and the coloring are correctly computed. Property 1 is shown to be correct with probability $1-\frac{1}{n}$ (refer to [17]). Furthermore, by Lemma 1 and Corollary 1, we have known that in Stages 1 and 2, the connected dominating set and the coloring is correctly computed with probability $1-O(n^{-1})$. Thus with probability $1-O(n^{-1})$, any message m can be broadcast to all nodes after the event $arrive(m)$ occurs for $O(D+k+\log^2 n)$ timeslots.

5 Lower Bound

In this section we give a lower bound on the time needed for a uniform randomized distributed algorithm to accomplish multiple-message broadcast. Recall that a randomized algorithm is called uniform if all awake nodes transmit a message with the same probability (independent of the communication history) in every timeslot. The proof of Theorem 2 can be found in [17].

Theorem 2. *Assume that all nodes use the same transmission power. Then, any uniform randomized multiple-message broadcast algorithm requires* $\Omega(D+k+\frac{\log^2 n}{\log\log\log n})$ *timeslots to disseminate all messages to the whole network with probability* $1-\frac{1}{n}$.

6 Conclusion

In this paper, assuming a practical network model for wireless ad-hoc and sensor networks as well as the SINR interference model, we propose the first randomized distributed multiple-message broadcast algorithm for networks with arbitrary message arrivals. In particular, we show that the proposed algorithm can disseminate any message m to the whole network in $O(D+k+\log^2 n)$ timeslots if there are at most k overlapping messages. We also show that any uniform randomized algorithm needs at least $\Omega(D+k+\frac{\log^2 n}{\log\log\log n})$ timeslots to accomplish multiple-message broadcast. Our algorithm outperforms all previous best known results [1,8], and breaks the $\Omega(k+D\log(n/D))$ lower bound [4,10] under the graph based radio network model. An important feature of the proposed algorithm is that, in contrast with the previous work, it does not need any neighboring information, e.g., an estimate on Δ. Based on the trivial $\Omega(D+k)$ lower bound, our algorithm is optimal for networks with diameter $D \in \Omega(\log^2 n)$. Therefore an obvious research direction is to design a faster multiple-message broadcast protocol for networks with diameter $D = o(\log^2 n)$. Another direction is to consider deterministic distributed algorithms for the multiple-message broadcast problem under the SINR model.

Acknowledgements. This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00302, the National Natural Science Foundation of China Grant 61103186, 61073174, 61033001, 61061130540, the Hi-Tech research and Development Program of China Grant 2006AA10Z216, and Hong Kong RGC-GRF grants 714009E and 714311.

References

1. Bar-Yehuda, R., Israeli, A., Itai, A.: Multiple communication in multi-hop radio networks. SIAM Journal on Computing 22, 875–887 (1993)
2. Censor-Hillel, K., Gilbert, S., Kuhn, F., Lynch, N.A., Newport, C.C.: Structuring unreliable radio networks. In: Proc. PODC (2011)
3. Chlebus, B., Gasieniec, L., Lingas, A., Pagourtzis, A.T.: Oblivious gossiping in ad-hoc radio networks. In: Proc. DIALM (2001)
4. Chlebus, B., Kowalski, D., Radzik, T.: Many-to-Many Communication in Radio Networks. Algorithmica 54(1), 118–139 (2009)
5. Goussevskaia, O., Moscibroda, T., Wattenhofer, R.: Local broadcasting in the physical interference model. In: Proc. DialM-POMC (2008)
6. Halldórsson, M.M., Mitra, P.: Optimal Bounds for Distributed Wireless Scheduling in the SINR Model. In: Proc. ICALP (2011)
7. Khabbazian, M., Kowalski, D.R.: Time-efficient randomized multiple-message broadcast in radio networks. In: Proc. PODC (2011)
8. Khabbazian, M., Kuhn, F., Kowalski, D.R., Lynch, N.A.: Decomposing broadcast algorithms using abstract MAC layers. In: Proc. DialM-POMC (2010)
9. Kuhn, F., Lynch, N.A., Newport, C.C.: The abstract MAC layer. Distributed Computing 24(3-4), 187–206 (2011)
10. Kushilevitz, E., Mansour, Y.: An $\Omega(D \log(n/D))$ lower bound for broadcast in radio networks. SIAM J. Comput. 27(3), 702–712 (1998)
11. 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)
12. Li, H., Hua, Q.-S., Wu, C., Lau, F.C.M.: Minimum-latency aggregation scheduling in wireless sensor networks under physical interference model. In: Proc. MSWiM (2010)
13. Li, H., Hua, Q.-S., Wu, C., Lau, F.C.M.: Latency-Minimizing Data Aggregation in Wireless Sensor Networks under Physical Interference Model. Ad Hoc Networks (to appear)
14. Scheideler, C., Richa, A., Santi, P.: An $O(\log n)$ dominating set protocol for wireless ad-hoc networks under the physical interference model. In: Proc. MOBIHOC (2008)
15. Woo, A., Culler, D.-E.: A transmission control scheme for media access in sensor networks. In: Proc. MOBICOM (2001)
16. Yu, D., Hua, Q.-S., Wang, Y., Lau, F.C.M.: An $O(\log n)$ Distributed Approximation Algorithm for Local Broadcasting in Unstructured Wireless Networks. In: Proc. DCOSS (2012)
17. Yu, D., Hua, Q.-S., Wang, Y., Tan, H., Lau, F.C.M.: Distributed Multiple-Message Broadcast in Wireless Ad-Hoc Networks under the SINR Model, http://iiis.tsinghua.edu.cn/~qshua/sirocco12full.pdf
18. Yu, D., Wang, Y., Hua, Q.-S., Lau, F.C.M.: Distributed $(\Delta + 1)$-Coloring in the Physical Model. In: Erlebach, T., Nikoletseas, S., Orponen, P. (eds.) ALGOSENSORS 2011. LNCS, vol. 7111, pp. 145–160. Springer, Heidelberg (2012)
19. Yu, D., Wang, Y., Hua, Q.-S., Lau, F.C.M.: Distributed local broadcasting algorithms in the physical interference model. In: Proc. DCOSS (2011)

Wireless Network Stability in the SINR Model

Eyjólfur Ingi Ásgeirsson, Magnús M. Halldórsson, and Pradipta Mitra

ICE-TCS
Reykjavik University
101 Reykjavik, Iceland
eyjo@hr.is, mmh@ru.is, ppmitra@gmail.com

Abstract. We study the stability of wireless networks under stochastic arrival processes of packets, and design efficient, distributed algorithms that achieve stability in the SINR (Signal to Interference and Noise Ratio) interference model.

Specifically, we make the following contributions. We give a distributed algorithm that achieves $\Omega(\frac{1}{\log^2 n})$-efficiency on all networks (where n is the number of links in the network), for all length monotone, sub-linear power assignments. For the power control version of the problem, we give a distributed algorithm with $\Omega(\frac{1}{\log n(\log n+\log\log \Delta)})$-efficiency (where Δ is the length diversity of the link set).

1 Introduction

We study the problem of scheduling packets over links in a wireless network, each link being a sender-receiver pair of wireless nodes. Since wireless signals propagate in all directions, simultaneous transmissions interfere with each other. This interference limits the number of transmissions that can succeed simultaneously. A wireless packet scheduling algorithm thus has to schedule packets efficiently, with respect to the limits imposed by interference.

In this setting, consider the following two related problems. The first: Given a set of links, how quickly (i.e., using how few slots) can all of the links be scheduled, taking interference into account? The second: Given is a set of links, and packets arrive at the senders of each link according to some stochastic process, where they remain queued until successfully transmitted to the receiver. Can one ensure that queue sizes at the senders remain bounded, in expectation? The first question is an "off-line" algorithmic problem, whereas the second comes from a queueing theoretic perspective where the input is probabilistic. In spite of their obvious commonalities, they are generally studied using quite disparate techniques. Our goal in this paper is to bridge a gap between these two related areas; specifically, to use recently developed algorithmic techniques to achieve results for the stochastic setting.

To do this, a crucial first step is to choose the right interference model – one that is faithful to physical reality yet is simple enough to be rigorously analyzable. In this paper, we adopt the SINR (Signal to Interference and Noise ratio)

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 123–134, 2012.

or *physical* model of interference. Compared to the more traditional and widely studied graph based models, the SINR model has been found to be realistic, and is enjoying increased attention and adoption [17,19,20]. This model (precisely defined in Section 2) is based on a realistic geometry of signal propagation (compared to unrealistic graph based interference models).

We are thus interested in algorithms that keep queue sizes bounded when faced with stochastic packet arrivals over arbitrary periods of time, assuming the SINR interference model. A network in which this goal is achieved is called *stable*. Stability has been an widely-studied metric for analyzing the performance of scheduling algorithms for wireless networks for quite some time. In a seminal work, Tassiulas and Ephremides [22], gave a characterization of those stochastic processes for which stability is possible in principle. The characterization is general enough to work for virtually any interference model. In light of this, the goal for the algorithm designer is to produce an (simple and efficient) algorithm that stabilizes networks for all (or a fair chunk) of these arrival processes. There is a long tradition of such work, e.g., [12,18], but they almost exclusively apply to graph-based interference models.

The Tassiulas-Ephremides characterization *is* formulated as a computational problem, which, if solved, would stabilize a network under all potentially stabilizable arrival rates. However, for the SINR model, this problem (known as the maximum weighted *capacity* problem) is NP-hard [1]. Not much is known about the algorithmic complexity of this problem (see [10] about a recent centralized result for linear power and more discussion about its relation to network stability), and almost nothing about possible distributed implementations. Thus, alternative approaches need to be sought.

In this work, we develop efficient and distributed scheduling algorithms for wireless network stability — by applying intuitions developed in recent research on the SINR model ([9] and [8] contain many references), all of which provide approximation algorithms to some relevant algorithmic questions.

One possible approach would be to apply algorithms for some of the core optimization problems as black boxes. For example, there are constant factor approximation algorithms for the capacity problem [11,9,14]. These alone are not sufficient, as there is no guarantee of fairness. Still, they can be easily turned into a $O(\log n)$-approximation for the weighted capacity problem. The problem with this approach, however, is that these algorithms are centralized, with no effective distributed algorithms in sight. Distributed algorithms are of crucial importance in the current setting. Our approach is therefore more of a "gray-box" one – while we adopt an algorithm of [15] as our basis, our analysis depends not on the overall approximation factor, but on more subtle properties of that algorithm.

Apart from being distributed, the main property a scheduling algorithm should have is high efficiency. "Efficiency" has a specific technical meaning which we define in Section 2. Intuitively, it captures how well the algorithm does compared to the Tassiulas-Ephremides characterization.

We achieve, depending on the algorithm chosen, efficiency ratios of $\Omega(\frac{1}{\log^2 n})$, $\Omega(\frac{1}{\log n})$ and $\Omega(\frac{1}{\log n(\log n+\log\log \Delta)})$, that are comparable or better than existing work on this topic. Our main algorithm requires only a "carrier sense" primitive to make it completely distributed. This is in contrast to many distributed algorithms in the literature (e.g., [18,16]) that are better described as "localized" – requiring an underlying infrastructure for wireless nodes to communicate with nearby nodes. This infrastructure, moreover, is usually not subject to the interference constraints of the original network. This is a rather strong assumption, especially in light of the fact that in a wireless network, one is presumably trying to establish such an infrastructure in the first place.

The paper is organized as follows. In Sections 2 and 3 we present the system model, our results, and more specific discussion on related work. In Section 4 we describe a general algorithmic framework for wireless scheduling. We then provide a specific instantiation of this framework that achieves good throughput performance for a large class of power assignments in general metric spaces, with implications for the power control case, where the power can be selected by links separately. Finally, in Section 5, we prove a more efficient, but centralized result for the power control case and present simulation results.

2 Model and Preliminaries

The SINR Model. The wireless network is modeled as a set L of n links, where each link $l \in L$ represents a potential transmission from a sender s_l to a receiver r_l, both points in a metric space. The distance between two points x and y is denoted $d(x, y)$. The distance from l''s sender to l's receiver is denoted $d_{l'l} = d(s_{l'}, r_l)$. The length of link l is denoted simply by $\ell = d(s_l, r_l)$.

The set may be associated with a *power assignment*, which is an assignment of a transmission power P_l to be used by each link $l \in L$. The signal received at point y from a sender at point x with power P is $P/d(x, y)^\alpha$ where the constant $\alpha > 0$ is the *path-loss exponent*.

We can now describe the *physical* or SINR-model of interference. In this model, a receiver r_l successfully receives a message from the sender s_l if and only if the following condition holds:

$$\frac{P_l/\ell^\alpha}{\sum_{l' \in S\setminus\{l\}} P_{l'}/d_{l'l}^\alpha + N} \geq \beta \ , \tag{1}$$

where N is the environmental noise, the constant $\beta \geq 1$ denotes the minimum SINR (signal-to-interference-noise-ratio) required for a message to be successfully received, and S is the set of concurrently scheduled links in the same *slot* (we assume that time is slotted.). We say that S is *SINR-feasible* (or simply *feasible*) if (1) is satisfied for each link in S.

A power assignment P is *length-monotone* if $P_v \geq P_w$ whenever $\ell_v \geq \ell_w$ and *sub-linear* if $\frac{P_v}{\ell_v^\alpha} \leq \frac{P_w}{\ell_w^\alpha}$ whenever $\ell_v \geq \ell_w$. This class includes the most interesting and practical power assignments, such that uniform power (all links use the same

power), linear power ($P_l = \ell^\alpha$, known to be energy efficient in the presence of noise), and mean power ($P_l = \ell^{\alpha/2}$, the assignment that produces maximum capacity in this class). We will also consider the "power control" case, where the power assignments are not predetermined, but have to be found out by the algorithm, and can be arbitrary.

Let $\Delta = \frac{\ell_{\max}}{\ell_{\min}}$, where $\ell_{\max}$ and $\ell_{\min}$ are, respectively, the maximum and minimum lengths in L.

Definition 1. *The* ***affectance*** $a_{l'}^P(l)$ *of link* l *caused by another link* l'*, with a given power assignment* P*, is the interference of* l' *on* l *relative to the power received, or*

$$a_{l'}^P(l) = \min\left\{1, c_v \frac{P_{l'}}{P_l} \cdot \left(\frac{\ell}{d_{l'l}}\right)^\alpha\right\} ,$$

where $c_v = \beta/(1 - \beta N \ell^\alpha / P_l)$.

The definition of affectance was introduced in [6] and achieved the form we are using in [15]. When clear from the context we drop the superscript P. Also, let $a_l^P(l) = 0$. Using the idea of affectance, Eqn. 1 can be rewritten as

$$a_S^P(l) \equiv \sum_{l' \in S} a_{l'}^P(l) \leq 1 ,$$

for all $l \in S$.

A link can schedule at most one packet during a slot, in other words, if a link has a queue, at most one packet from the queue can be scheduled during a single slot.

Stability of Stochastic Processes. We assume that packets arrive at the sender of each link l according to a stochastic process with average arrival rate m_l.

We define stability as such.

Definition 2. *An algorithm* ***stabilizes*** *a network for a particular arrival process if, under that arrival process the average queue size at each link is bounded (ie, does not grow asymptotically with time).*

The *throughput region* is then the set of all possible arrival rate vectors such that there exists some scheduling policy that can stabilize the network. As proved in [22], the throughput region is characterized by

$$\Lambda = \{\lambda : \lambda \preceq \phi, \text{for some } \phi \in Co(\Omega)\} ,$$

here Ω is the set of vectors in $\mathbb{R}^n$ characterizing all maximal feasible sets (i.e., each vector in Ω is a binary vector with 1's in indices corresnponding to links belonging to the relevant maximal feasible set). $Co(\Omega)$ is the convex hull of Ω; λ and ϕ are vectors in $\mathbb{R}^n$, indicating arrival rates on links, and $\lambda \preceq \phi$ means that each entry of λ is less than or equal to the corresponding entry in ϕ.

In the best case, one would like stabilize all of Λ. If that is not possible, the hope is to achieve a high *efficiency ratio*:

Definition 3. *The* ***efficiency ratio*** γ *of a scheduling algorithm is* $\gamma = \sup\{\gamma :$ *all networks are stabilized for all* $\lambda \in \gamma\Lambda\}$*. The algorithm is then* γ***-efficient****.*

We assume that the arrival processes are independent accross time (and links). For a certain efficiency γ, for all permissible arrival rate vectors λ, $\lambda \preceq \sum_i m_i M_i$ where each M_i is a maximal feasible set, and m_i are weights such that $\sum_i m_i = \gamma$. Let the expected arrival rate on a link l be m_l, it can be easily seen that

$$m_l = \sum_{i:l\in M_i} m_i \leq \gamma \ . \tag{2}$$

3 Results and Related Work

Our main results are:

Theorem 1. *For all given networks with links on metric spaces, and all sublinear, length-monotone power assignments, there exists a* $\Omega(\frac{1}{\log^2 n})$*-efficient distributed algorithm.*

Theorem 2. *For all given networks with links on the Euclidean plane, there exists a* $\Omega\left(\frac{1}{\log n(\log n+\log\log \Delta)}\right)$*-efficient distributed power control algorithm. A centralized algorithm exists that achieves* $\Omega\left(\frac{1}{\log n}\right)$ *efficiency.*

We are aware of two earlier papers on stability in the SINR model. In [16], the authors study the Longest Queue First algorithm (a classical algorithm that can be seen as a natural extension of maximal weighted matching). They show that LQF is not stable, but a variant works well. A "localized" implementation is provided, i.e., it is shown that the algorithm can be implemented in a distributed manner if links can communicate with other "neighboring" links arbitrarily. The achieved efficiency ratio in $\Omega(\frac{1}{\Delta^\alpha})$ (the dependence on Δ is not explicitly mentioned, but can be seen to be necessary). Our recent paper [2] is a companion of the current work, where an extremely simple and completely distributed algorithm achieving $\Omega(\frac{1}{\Delta^\alpha})$ efficiency is introduced. In comparison, the results in the current work involve efficiency that is logarithmically dependent on n (and, in one case, doubly logarithmic in Δ). Dependence on n and Δ are theoretically not comparable, and either could be preferable in practice. The distributed algorithm in this paper has to assume a carrier-sense primitive, which is not assumed in [2]. However, it does not need to have a special communication infrastructure with neighboring links.

The body of work on wireless network stability in *other* models is too vast to survey properly. In terms of efficiency ratio, a range of results have been derived in a variety of models. Naturally one seeks efficiency of 1 [18] whenever possible, but results for efficiency ratios of 1 under certain conditions [4], or $\frac{1}{6}$ [12] can be found in the literature. Ratios in terms of certain network characteristics are known as well [12,16]. For the SINR model, which is being studied only

very recently, an efficiency ratio of a constant that is independent of network parameters is not known.

Technically, we depend heavily on [15] that provides a $O(\log^2 n)$ approximation algorithm for the scheduling problem. The algorithm and technical aspects of this work used here will be introduced in the following section as needed.

We are aware of a very recent unpublished work of Kesselheim [13] achieving results in the SINR model very similar to the present paper.

4 Main Algorithm

The basic algorithmic framework used is listed as **General** below. For simplicity, we treat it as a centralized procedure first and discuss distributed implementations later.

Algorithm 1. General(θ, $\mathcal{A}$)

```
The algorithm maintains a FIFO queue S of sets, such that each S ∈ S is feasible.
At the beginning, S ← ∅.
for time t ← 1, 2, ... do
   if S is non-empty then
      Schedule the first S ∈ S
      S ← S − S
   end if
   if t mod θ = 1 then
      Let L = new packet arrivals in the time period [t − θ, t − 1]
      q = t/θ
      Use algorithm A to find a schedule R_q = ∪_j R'_j for L
      Append R_q (in any order) to S
   end if
end for
```

The algorithm takes two parameters. One is θ, a number that defines the "period" of the algorithm. The second parameter is an algorithm $\mathcal{A}$ which can solve the *scheduling problem*, used as a black box by **General** to compute schedules. The *scheduling problem* is the optimization problem where given a set L of links, one seeks to partition L into minimum number of sets such that each of these sets is feasible (i.e., can be transmitted in one slot). Since the problem is NP-hard, we will work with approximation algorithms. Depending on the result we seek, we will set θ and $\mathcal{A}$ accordingly.

General can be alternatively described in the following way. The algorithm divides the time slots into consecutive periods of length θ each. Let us denote these periods as $C_1, C_2 \ldots$ etc. At the beginning of period C_q, the algorithm computes a schedule $\mathcal{R}_{q-1}$ of the links produced in C_{q-1}. It does so using $\mathcal{A}$. **General** then adds these computed feasible sets to the set $\mathcal{S}$. Now during each slot of C_q, the algorithm schedules the first set from $\mathcal{S}$ (which is implemented

as a FIFO queue). It does this until C_q ends, in which case it moves on to the next period, or until $\mathcal{S}$ is empty, in which case it waits until the end of C_q. Note that there is nothing to schedule during C_1, we just wait during this time.

Let Q_l^t be the queue length at link l at time t. First, note that $Q_l^t \leq S^t$ for all l, where $S^t = |\mathcal{S}|$ at time t ($\mathcal{S}$ is as in the algorithm **General**). This is the number of slots we require to schedule all links outstanding at time t. Obviously, one cannot schedule all links in time less than the size of the longest queue (since copies of the same link cannot be scheduled together). Thus a bound on S^t immediately gives us a bound on Q_l^t (for all l). Consequently, from now on we will focus on bounding S^t. Also note that it suffices to bound S^t on period boundaries, i.e., at times t such that $t \bmod \theta = 1$. This is because, in expectation, the queue lengths cannot grow by much during the course of a period. Let ρ be a large enough constant.

For simplicity, we will assume that the arrival distributions at every link l is a Bernoulli random variable with mean m_l. To prove Theorem 1, we set $\theta = 10\rho^2 \log^2 n$.

4.1 The Scheduling Algorithm

We also select as $\mathcal{A}$ the scheduling algorithm described in [15]. It is known [8] that this algorithm achieves a $O(\log n)$-approximation factor to the scheduling problem. We are, however, interested in a slightly different performance measure of the algorithm in [15]. For a link set R, define the *maximum average affectance* $\bar{A}(R) = \max_{Q \subseteq R} \frac{1}{|Q|} \sum_{l \in Q} \sum_{l' \in Q} a_l(l')$. It is known that:

Theorem 3. *[15] The algorithm $\mathcal{A}$ has expected running time of at most $\rho \log n \cdot \bar{A}(R)$ on a link set R.*

We now turn our attention to proving the stability of the algorithm, i.e., Thm. 1. Given the efficiency claimed in Thm. 1, it sufficient to deal with stochastic processes satisfying

$$\sum_{M_i} m_i \leq \frac{1}{5\rho^2 \log^2 n} \ . \tag{3}$$

Lemma 1. $\mathbb{E}(|\mathcal{R}_q|) \leq O(\log^2 n) < \theta$ *for all* q.

Proof. Define $a^{t+}(l)$ to be the outgoing affectance from a link l to longer links appearing in slot t. Formally, if $X_l(t)$ is the Bernoulli random variable denoting the number of packets that arrived at the sender of link l during slot t, then

$$a^{t+}(l) \equiv \sum_{l' \in L, l' \geq l} a_l(l') X_{l'}(t) \ . \tag{4}$$

Let $A^+(l)$ be the sum of $a^{t+}(l)$ over all θ slots in the period, or,

$$A^+(l) = \sum_{t=1}^{\theta} a^{t+}(l) \ .$$

Now, we claim,

Claim. $\mathbb{E}(A^+(l)) \leq \frac{\theta}{5\rho \log n}$.

Proof. In [15], it is shown that for any feasible set M_i

$$\sum_{l' \in M_i, l' \geq l} a_l(l') \leq \rho \log n \ . \tag{5}$$

Thus, for any time slot t,

$$\mathbb{E}(a^{t+}(l)) \overset{1}{=} \sum_{l' \in L, l' \geq l} a_l(l') \mathbb{E}(X_{l'}(t)) \overset{2}{=} \sum_{l' \geq l} a_l(l') m_{l'} \overset{3}{\leq} \sum_{l' \geq l} a_l(l') \sum_{i : l' \in M_i} m_i$$
$$\overset{4}{=} \sum_i m_i \sum_{l' \in M_i, l' \geq l} a_l(l') \overset{5}{\leq} \rho \log n \sum_i m_i \leq \frac{1}{5\rho \log n} \ ,$$

with explanations of numbered (in)equalities:

1. By definition of $a^{t+}(l)$ (Eqn. 4)
2. $\mathbb{E}(X_{l'}(t)) = m_{l'}$ by definition, since they both express the expected number of packets arriving in each time slot on l'.
3. By Eqn. 2.
4. Rearrangement.
5. By Eqn. 5.
6. By Eqn. 3.

Now, by the Chernoff-Hoeffding inequality (see, for example, Eqn. 1.8, Thm. 1.1 of [5]):

$$\mathbb{P}(A^+(l) \geq r) \leq \frac{1}{2^r} \ ,$$

for all $r \geq \frac{2\theta}{5\rho \log n} = 4\rho \log n$. Defining $A^+_{\max} = \max_l A^+(l)$, and union bounding we get,

$$\mathbb{P}(A^+_{\max} \geq r) \leq \frac{n}{2^r} \ . \tag{6}$$

We analogously define $a^{t-}(l) \equiv \sum_{l' \in L, l' \geq l} a_{l'}(l) X_{l'}(t)$ to be the incoming affectance from longer links. We similarly define A^-_{max}, and obtain that $\mathbb{P}(A^-_{\max} \geq r) \leq \frac{n}{2^r}$, which depends on the bound

$$\sum_{l' \in M_i, l' \geq l} a_{l'}(l) \leq \rho \ , \tag{7}$$

also proven in [15]. (Note that the bound here is tighter than Eqn. 5).

It is not hard to verify that that $\bar{A} \leq A^+_{\max} + A^-_{\max}$. Thus,

$$\mathbb{E}(\bar{A}) \leq \mathbb{E}(A^+_{\max}) + \mathbb{E}(A^-_{\max}) \leq 2\mathbb{E}(A^+_{\max})$$
$$\leq \frac{2\theta}{5\rho \log n} + \sum_{i=1}^{\infty} 2^i \frac{2\theta}{5\rho \log n} \frac{n}{2^{2^i \frac{2\theta}{5\rho \log n}}}$$
$$\leq \frac{2\theta}{5\rho \log n} + \frac{2\theta}{5\rho \log n} = \frac{4\theta}{5\rho \log n} \ .$$

Now, by Theorem 3, $\mathbb{E}_{\mathcal{A}}(|\mathcal{R}_q|) \leq \rho \log n \cdot \bar{A}$, where $\mathbb{E}_{\mathcal{A}}$ denotes expectation over the random bits of the algorithm (which is randomized). Noting that the random bits of the algorithm are independent of the arrival process, we can use the bound on $\mathbb{E}(\bar{A})$ to claim that

$$\mathbb{E}(|\mathcal{R}_q|) \leq \rho \log n \cdot \frac{4\theta}{5\rho \log n} < \theta \ .$$

Now we can prove Thm. 1. Note that by Lemma 1, the expected scheduling cost required for packets produced during a single period ($\mathbb{E}(|R_q|)$) is strictly smaller than the scheduling capacity of a single period (θ). With this observation, we can reduce our system to a very basic queueing system:

- A single server, an infinite queue, and slotted time. The time slots in this system correspond to the periods of the original system.
- At the beginning of each slot a fixed number s of packets are served and leave the system (s corresponds to θ).
- At the end of every slot, a random number of new packets arrive. This is a random variable A on the non-negative integers, and $\mathbb{E}(A) = a$ (a corresponds to $\mathbb{E}(|\mathcal{R}_q|)$).

Now, if $a < m$, the corresponding countable Markov chain has a stationary distribution, and if a is square integrable, the expected queue length will be finite (see [3], for example). The condition $a < m$ is easily seen to be true (since $\mathbb{E}(|\mathcal{R}_q|) < \theta$ by Lemma 1). Square integrability of $|\mathcal{R}_q|$ follows from the fact that $\mathcal{R}_q$ admits a large deviation bound (this is implicit in the proof of Lemma 1). Of course the "queue length" of this system corresponds to S^t, the number of slots the outstanding packets would require to be scheduled. Thus we have proven $\mathbb{E}(S^t)$ to be bounded, and as observed before, this is enough to complete the proof of the theorem.

4.2 Implications for the Power Control Problem

It was shown in [7,9] that *mean power* (where P_l is set to $\ell^{\alpha/2}$) achieves a good approximation to the power control problem. Since Thm. 1 covers the mean power assignment, this gives us a distributed algorithm for the power control problem. To achieve the bound claimed in Thm. 2 for the distributed algorithm, the main ingredient is the following bound (analogous to Eqn. 5).

Lemma 2 ([7,9]). *If M is a feasible set with respect to* any *power assignment,*

$$\sum_{l' \in M, l' \geq l} a_{l'}(l) \leq c_3(\log n + \log\log \Delta) \ , \tag{8}$$

where the affectance $a_{l'}(l)$ is measured using mean power.

With this bound (and a similar analog of Eqn. 7) in hand, the proof technique of Thm. 2 can be duplicated to achieve the bound claimed in Thm. 2 for distributed algorithms.

4.3 Distributed Implementation

We now demonstrate how to implement **General** in a distributed fashion. Interestingly, we require very few additional assumptions to make this work. The basic tool is the algorithm in [15], listed as **Distr-SingleLink**.

Algorithm 2. Distr-SingleLink

```
1: k ← 0
2: while transmission not successful do
3:    q = 1/(4·2^k)
4:    for 8 ln n / q slots do
5:       transmit with i.i.d. probability q
6:    end for
7:    k ← k + 1
8: end while
```

The first thing to note here is that **Distr-SingleLink** itself is completely distributed. In other words, **General**, if applied to the links produced during a single period, could be implemented in a distributed manner straightaway. The challenge is that our bounds assume that the algorithm works in a FIFO manner, thus **Distr-SingleLink** for a packet produced in C_q should not start executing until all packets produced up to C_{q-1} have been successfully scheduled.

To implement this, we assume that each sender in the system maintains a few counters. The first counter *cur* keeps track of the current period, and a second counter s tracks the current period being *scheduled* (naturally $s \leq cur$). For each outstanding packet p in the queue, the sender also maintains the period in which it was generated (g_p). The counter *cur* is easily maintained, by incrementing it once every θ slots. The third counter is equally simple, when a packet p arrives, g_p is assigned the current value of *cur*. We will describe how s is maintained below, but note that given s, the algorithm can now be easily implemented in a distributed fashion. For each packet p, the sender waits until $s = g_p$, and then runs **Distr-SingleLink** for p until the link successfully transmits.

Maintaining s is slightly more, but not too, involved. It is here that we need to make an additional assumption, which is that nodes (senders) can sense the channel to determine transmission activity. This is a not uncommon assumption (see, e.g., [21]), based on the Clear Channel Assessment capability in the 802.11 wireless standard.

Let us divide the time slots into consecutive pairs. The first time slot is used for normal transmission (i.e., executing **General** and **Distr-SingleLink**). The second slot is used for signaling. Senders that are transmitting currently (i.e., senders that have at least one non-transmitted packet p for which $g_p = s$) use the signaling slot to simply signal that they have still not completed. Thus when all links from period s succeed, a silent signaling slot appears. All senders register this event by sniffing the channel, and increment s.

From a practical point of view, wasting every other slot for signaling, as well as assuming some sort of "perfect" carrier sensing capability, is problematic. Simulation studies presented are done with practically reasonable approximations to these assumptions (indeed, these heuristics seem to help the algorithm).

5 Extensions and Simulations

In the power control version of the problem, selecting (an arbitrary) power for each link is a part of the problem. Feasible sets are now those for which there exists an (unknown) power assignment that allows for simultaneous transmissions of all the links.

Using a centralized algorithm of [14], a $O(\frac{1}{\log n})$ throughput is achievable. We omit details.

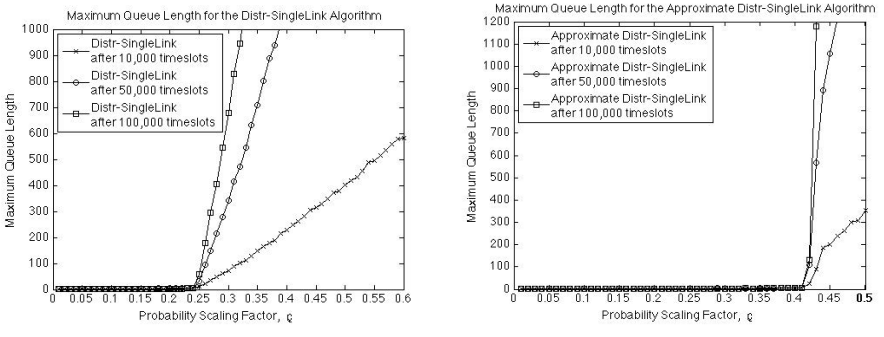


Fig. 1. The maximum queue lengths for the distributed algorithm Distr-SingleLink. The problem instances are based on random topology with $n = 200$, $\ell_{\min} = 1$, $\ell_{\max} = 20$, $\alpha = 2.5$ and $\beta = 1$.

We implemented **General** (using **Distr-SingleLink**) on a random topology. As Fig. 1 shows, efficiency ratios upto 0.4 is achieved, which is quite good. Details are omitted due to space restrictions.

References

1. Andrews, M., Dinitz, M.: Maximizing capacity in arbitrary wireless networks in the SINR model: Complexity and game theory. In: INFOCOM, pp. 1332–1340 (2009)
2. Ásgeirsson, E.I., Halldórsson, M.M., Mitra, P.: A Fully Distributed Algorithm for Throughput Performance in Wireless Networks. In: CISS (2012)
3. Asmussen, S.: Applied Probability and Queues, 2nd edn. Springer (2003)
4. Dimakis, A., Walrand, J.: Sufficient conditions for stability of longest-queue-first scheduling: second-order properties using fluid limits. Advances in Applied Probability 38(2), 505–521 (2006)
5. Dubhashi, D.P., Panconesi, A.: Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press (2009)

6. Goussevskaia, O., Halldórsson, M.M., Wattenhofer, R., Welzl, E.: Capacity of Arbitrary Wireless Networks. In: INFOCOM, pp. 1872–1880 (April 2009)
7. Halldórsson, M.M.: Wireless Scheduling with Power Control. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 361–372. Springer, Heidelberg (2009)
8. 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, Part II. LNCS, vol. 6756, pp. 625–636. Springer, Heidelberg (2011)
9. Halldórsson, M.M., Mitra, P.: Wireless Capacity with Oblivious Power in General Metrics. In: SODA (2011)
10. Halldorsson, M.M., Mitra, P.: Wireless capacity and admission control in cognitive radio. In: INFOCOM (2012)
11. Halldórsson, M.M., Wattenhofer, R.: Wireless Communication Is in APX. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 525–536. Springer, Heidelberg (2009)
12. Joo, C., Lin, X., Shroff, N.: Understanding the Capacity Region of the Greedy Maximal Scheduling Algorithm in Multi-Hop Wireless Networks. In: INFOCOM (2008)
13. Kesselheim, T.: Dynamic packet scheduling in wireless networks, http://arxiv.org/abs/1203.1226
14. Kesselheim, T.: A Constant-Factor Approximation for Wireless Capacity Maximization with Power Control in the SINR Model. In: SODA (2011)
15. 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)
16. Le, L.B., Modiano, E., Joo, C., Shroff, N.B.: Longest-queue-first scheduling under SINR interference model. In: MobiHoc (2010)
17. Maheshwari, R., Jain, S., Das, S.R.: A measurement study of interference modeling and scheduling in low-power wireless networks. In: SenSys, pp. 141–154 (2008)
18. Modiano, E., Shah, D., Zussman, G.: Maximizing throughput in wireless networks via gossiping. In: SIGMETRICS/Performance, pp. 27–38 (2006)
19. Moscibroda, T., Wattenhofer, R., Weber, Y.: Protocol Design Beyond Graph-Based Models. In: Hotnets (November 2006)
20. Moscibroda, T., Wattenhofer, R., Zollinger, A.: Topology control meets SINR: The scheduling complexity of arbitrary topologies. In: MobiHoc, pp. 310–321 (2006)
21. Scheideler, C., Richa, A.W., Santi, P.: An $O(\log n)$ dominating set protocol for wireless ad-hoc networks under the physical interference model. In: MobiHoc, pp. 91–100 (2008)
22. Tassiulas, L., Ephremides, A.: Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. IEEE Trans. Automat. Contr. 37(12), 1936–1948 (1992)

What Can Be Computed without Communications?

Heger Arfaoui and Pierre Fraigniaud*

LIAFA, CNRS and University Paris Diderot, France

Abstract. This paper addresses the following 2-player problem. Alice (resp., Bob) receives a boolean x (resp., y) as input, and must return a boolean a (resp., b) as output. A *game* between Alice and Bob is defined by a pair (δ, f) of boolean functions. The objective of Alice and Bob playing game (δ, f) is, for every inputs x and y, to output values a and b, respectively, satisfying $\delta(a, b) = f(x, y)$, in *absence of any communication* between the two players.It is known that, for XOR-games, that is, games equivalent, up to individual reversible transformations, to a game (δ, f) with $\delta(a, b) = a \oplus b$, the ability for the players to use entangled quantum bits (qbits) helps: there exist a distributed protocol for the CHSH game, using quantum correlations, for which the probability that the two players produce a successful output is higher than the maximum probability of success of any classical distributed protocol for that game, even when using shared randomness.

In this paper, we show that, apart from XOR-games, quantum correlations does not help, in the sense that, for every such game, there exists a classical protocol (using shared randomness) whose probability of success is at least as large as the one of any protocol using quantum correlations. This result holds for both worst case and average case analysis. It is achieved by considering a model stronger than quantum correlations, the *non-signaling model*, for which we show that, if the game is not an XOR-game, then shared randomness is a sufficient resource for the design of optimal protocols. These results provide an invitation to revisit the theory of distributed *checking*, a.k.a. distributed *verification*. Indeed, the literature dealing with this theory is mostly focusing on decision functions δ equivalent to the AND-operator. This paper demonstrates that such a decision function may not well be suited for taking benefit of the computational power of quantum correlations.

1 Introduction

1.1 Context and Objective

This paper addresses the following 2-player problem. Alice (resp., Bob) receives a boolean x (resp., y) as input, and must return a boolean a (resp., b) as output.

* Both authors are supported by the ANR projects DISPLEXITY and PROSE, and by the Interdisciplinary project "Algorithmique distribuée quantique" of University Paris Diderot. Additional support from the INRIA project GANG.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 135–146, 2012.

A *game* between Alice and Bob is defined by a pair (δ, f) of boolean functions. The objective of Alice and Bob playing game (δ, f) is, for every inputs x and y, to output values a and b satisfying

$$\delta(a, b) = f(x, y)$$

in *absence of any communication* between the two players. Obviously, the game is trivial whenever there exist two boolean functions α and β such that $\delta(\alpha(x), \beta(y)) = f(x, y)$ for every pair $(x, y) \in \{0, 1\}^2$. Indeed, for such games, there exists a deterministic distributed protocol solving the game, with Alice returning $\alpha(x)$ on input x, and Bob returning $\beta(y)$ on input y. Non-trivial games may still be solved, but only under some probabilistic guarantees. A game (δ, f) is said to be solvable with probability p if there exists a randomized distributed protocol such that Alice outputs a, and Bob outputs b, with

$$\Pr(\delta(a, b) = f(x, y)) \geq p \tag{1}$$

for every input pair $(x, y) \in \{0, 1\}^2$.

Different sources of randomness can then be considered. Classical[1] sources of randomness include the case where each of the two players are provided with individual independent sources of random bits. It also include shared randomness where, in addition to individual independent sources of random bits, the two players have access to a common source of random bits. Shared randomness enables to produce outputs satisfying

$$\Pr(a, b|x, y) = \sum_{\lambda \in \Omega} \Pr(a|x, \lambda) \cdot \Pr(b|y, \lambda) \cdot \Pr(\lambda) \tag{2}$$

where the random variable λ is drawn from some probability space Ω, and $\Pr(a, b|x, y)$ denotes the probability that Alice outputs a and Bob outputs b, given the fact that Alice receives x as input, and Bob receives y as input. It is known [3] that correlations on quantum entangled states enable to derive protocols whose output distribution cannot be modeled as Eq. 2. One evidence of this fact is the CHSH game [6]:

$$a \oplus b = x \wedge y$$

where $\oplus$ denotes the exclusive-or operator. CHSH can be solved with probability $\cos^2(\pi/8) > \frac{3}{4}$ with a quantum protocol [5], while every protocol using classical shared randomness cannot solve CHSH with probability more than $\frac{3}{4}$. One objective of this paper is to complete an exhaustive study of 2-player games in order to identify for which games quantum correlations help.

In fact, the literature dealing with 2-player games (see, e.g., [1,2,8,15], and the recent survey [4]) refers to objects called *boxes*. A box B is characterized by the probabilities $\Pr(a, b|x, y)$ of outputting pair (a, b) given the input pair (x, y), for all $a, b, x, y \in \{0, 1\}$. A box B is thus described by a set

$$\{\Pr(a, b|x, y),\ (x, y) \in \{0, 1\}^2\}$$

[1] I.e., not using quantum effects.

of four probability distributions, one for each pair $(x, y) \in \{0, 1\}^2$. Hence, there are infinitely many boxes, with different computational powers.

The absence of communication between the two players along with the assumption of causality are captured by the class of *non-signaling* boxes. A box B is non-signaling if and only if it satisfies that the marginal output distributions for Alice and Bob depend only on their respective inputs. Formally, a non-signaling box satisfies:

$$\begin{aligned} \forall a, x,\ \textstyle\sum_b \Pr(a, b|x, 0) = \sum_b \Pr(a, b|x, 1), \\ \text{and}\ \ \forall b, y,\ \textstyle\sum_a \Pr(a, b|0, y) = \sum_a \Pr(a, b|1, y) \end{aligned} \tag{3}$$

Non-signaling boxes satisfying Eq. 2 are called *local*, where "locality" is referring here to the physical science concept of *local hidden variables* [3,9]. Boxes that do not satisfy Eq. 3 are *signaling*. Signaling boxes are not considered physically realistic because they would imply instantaneous transmission of signals between two distant entities.

The set of all boxes has a geometric interpretation [1], for it forms a 12-dimensional convex polytope, including the (convex) polytope of non-signaling boxes, which includes in turn the (convex) local polytope. Fig. 1 provides an abstract representation of the non-signaling polytope. Each of the extremal vertices of the non-signaling polytope is equivalent (up to individual reversible transformations on the inputs and outputs) to the PR box [5,15], that is described by the distribution:

$$\Pr(a, b|x, y) = \begin{cases} \frac{1}{2} \text{ if } a \oplus b = x \wedge y \\ 0 \text{ otherwise.} \end{cases}$$

Notice that the PR box satisfies $\Pr(a \oplus b = x \wedge y) = 1$ for every input pair x, y. So, in particular, it solves the CHSH game with probability 1. Each of the extremal vertices of the local polytope can be implemented by a deterministic protocol: they are equivalent to the identity box ID described by $\Pr(a, b|x, y) = 1$ if and only if $a = x$ and $y = b$. Every non-extremal box B is a linear combinations of extremal boxes: $B = \sum_{i=1}^{k} \beta_i B_i$ where B_i is an extremal box, $\sum_{i=1}^{k} \beta_i = 1$, and $\beta_i > 0$ for every $i = 1, \ldots, k$. On Fig. 1, the doted line represents the limit of the class of boxes implementable by a quantum protocol. This latter class strictly contains the local boxes, and is strictly included in the class of non-signaling boxes, as witnessed by the CHSH game.

Our objective can thus be reformulated as follows. Given a box implementable by a quantum protocol, which games can be efficiently solved using this box? Stated differently, given a game, what are the boxes implementable by a quantum protocol that enable to solve that game with better guarantees than any local boxes?

1.2 Our Results

We show that, for every 2-player game (δ, f) different from an XOR-game, i.e., different from a game which is equal, up to individual reversible transformations,

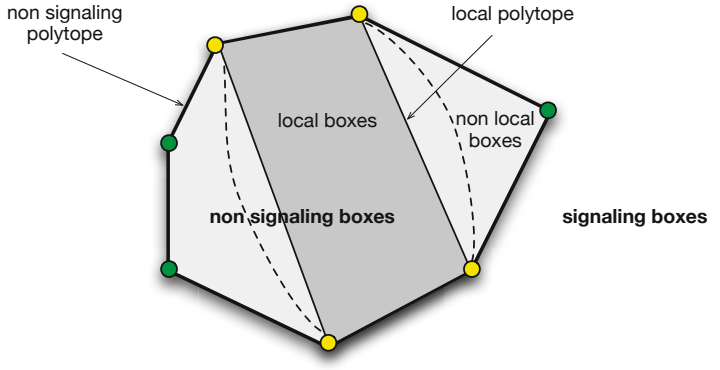


Fig. 1. Abstract representation of the non-signaling polytope

to a game (δ, f) with $\delta(a, b) = a \oplus b$, every box solving (δ, f) with probabilistic guarantee p greater that the probabilistic guarantee of any local box, is signaling. As a corollary, quantum correlations do not help for solving games different from XOR-games. Moreover, this result holds even the worst-case guarantee stated in Eq. 1 is replaced by the average-case guarantee

$$\frac{1}{4}\sum_{x,y} \Pr(\delta(a,b) = f(x,y)) \geq p \ .$$

The results in this paper open new perspectives in term of distributed *checking*, a.k.a. distributed *verification*, which consists in having a set of, say, n processes deciding whether their global state (defined as the union of the local state of every individual process) satisfies some prescribed property, or not. The literature on this latter topic (see, e.g., [7,10,11,13,14]) assumes a *decision* function δ which is applied to the set of individual decisions produced by the processes. Typically, each process should output a boolean b_i, and the global interpretation of the outputs is computed by

$$\delta(b_1, \ldots, b_n) = \bigwedge_{i=1}^{n} b_i \in \{\text{“yes”}, \text{“no”}\} \ .$$

The use of the AND operator is motivated by the requirement that the global state is valid if and only if all processes agree on some (local) validity condition. If this condition is locally violated somewhere in the system, then at least one process “rises an alarm” by outputting 0. However, recent advances in the theory of distributed checking [12] demonstrate that using other decision functions δ significantly increases the power of the “checker”, or “verifier”. Our results show that some functions δ, in particular the classical AND operator, do not enable to

use the power of quantum computing efficiently, compared to shared randomness, at least for 2-player games. In contrast, the exclusive-or operator is known to offer high potential, as far as distributed quantum computing is concerned. In particular, [2] proved that every boolean function f on n independent players can be implemented by a circuit of PR boxes that output booleans b_i, $i = 1, \ldots, n$, satisfying

$$\bigoplus_{i=1}^{n} b_i = f(x_1, \ldots, x_n) \ .$$

The results in this paper give one more evidence of the impact of the decision function δ on the ability of "deciding" boolean predicates f.

2 Equivalence Classes of Games

As introduced in the previous section, a game between Alice and Bob is described by a pair (δ, f) of boolean functions on two variables. Playing the game means for Alice (resp. Bob) to receive a boolean x (resp., y) as input, and to return a boolean a (resp., b) as output such that $\delta(a, b) = f(x, y)$ without communication between the two players. Examples of games are

$$\text{EQ}: \quad a \wedge b = \overline{x \oplus y} \quad \text{and} \quad \text{NEQ}: \quad a \wedge b = x \oplus y.$$

Another example of a game is :

$$\text{AMOS}: \quad a \wedge b = \overline{x \wedge y}.$$

In these three examples, one can view the games as Alice and Bob respectively deciding whether the equality $x = y$ holds, whether the non-equality $x \neq y$ holds, and whether there is "at most one selected" player (a selected player has input 1). Here, "deciding" means that if the answer is "yes" then both players should output "yes", while if the answer is "no" then at least one player should output "no". In fact, the three games EQ, NEQ, and AMOS, are AND-games. However, all games are not of that type. In particular, we shall see that the already mentioned CHSH game

$$a \oplus b = x \wedge y$$

is not an AND-game, for $\delta(a, b) \neq a \wedge b$. More precisely, for any game (δ, f), both functions δ and f can be rewritten as:

$$\delta(a, b) = \alpha_{1,1}ab + \alpha_{1,0}a + \alpha_{0,1}b + \alpha_{0,0} \ \text{ and } \ f(x, y) = \beta_{1,1}xy + \beta_{1,0}x + \beta_{0,1}y + \beta_{0,0}$$

where the $+$ symbol denotes the exclusive-or operator $\oplus$, the (omitted) $\cdot$ symbol denotes the and-operator $\wedge$, and all coefficients are in $\{0, 1\}$. We say that two games (δ, f) and (δ', f') are equivalent if

$$\delta(a, b) = \delta'(A, B) \ \text{ and } \ f(x, y) = f'(X, Y)$$

where A (resp., B, X, Y) is a degree-1 polynomial in a (resp., b, x, y) with coefficients in $\{0, 1\}$. Whenever two games are equivalent, any protocol solving one of the two games can be used for solving the other games, by performing individual reversible transformations on the inputs and outputs, and the probability of success for the two games will be identical. The same notion of equivalence can be defined for boxes. Now we can state formally that the CHSH game is not equivalent to any of the three AND-games: EQ, NEQ, or AMOS. This is because, as we will see further in the text, none of these latter games can be solved with probability 1 by a non-signaling box (as opposed to the CHSH game which can be solved with probability 1 by the PR box). Instead, EQ and NEQ are equivalent games. Indeed, for NEQ, $f(x, y) = x + y$, while, for EQ, $f(x, y) = x + (y + 1)$.

Definition 1. *A game (δ, f) is an* XOR*-game if and only if it is equivalent to a game (δ', f') where $\delta'(a, b) = a \oplus b$.*

3 On the Power of Quantum Correlations

In this paper, we establish our main result, stating that correlations on quantum entangled states do not help for solving 2-player games that are not equivalent to an XOR-games. In fact we establish a stronger result by showing that non-signaling boxes do not help for those games, compared to local boxes.

Theorem 1. *Let (δ, f) be a 2-player game that is not equivalent to any* XOR*-game. Let p be the largest success probability for (δ, f) over all local boxes. Then every box solving (δ, f) with probabilistic guarantee $> p$ is signaling.*

Proof. The proof is straightforward for games (δ, f) where δ does not depend on both a and b. Indeed, if δ is constant, say α, then the game is either impossible (whenever $\exists x, y : f(x, y) \neq \alpha$) or trivial (whenever $\forall x, y, f(x, y) = \alpha$). And if δ is a single-variable function, say $\delta(a, b) = a + \alpha$ for some α, then the game is again either impossible, or trivial, or equivalent to a single-player game where the player must compute a two-variable function $f(x, y)$ knowing only one of the variables. Games of that latter class are equivalent to either the game $a = y$ or the game $b = x$. Non-signaling boxes do not help for such games (the best probability of success is $\frac{1}{2}$). Therefore, we focus now on "true" 2-player games, i.e., games (δ, f) where δ depends on both a and b.

First, we show that every true 2-player game (δ, f) which is not equivalent to an XOR-game is either deterministic, or equivalent to NEQ or AMOS. To establish this claim, observe that if f is constant, or depends on only one of the the two inputs, then the game (δ, f) can be solved with probability 1, by a deterministic protocol. Indeed, assume, without loss of generality, that f depends only on x. (The case f constant is straightforward). Then Alice and Bob can agree beforehand on a fixed value b^* for b. It follows that, knowing b^*, f ,and δ, Alice can output a such that $\delta(a, b^*) = f(x)$.

We can now come to the interesting case, that is, when both δ and f depend on their two inputs. Any 2-variable boolean function g can be rewritten as :

$$g(u, v) = U + V \quad or \quad g(u, v) = UV \quad or \quad g(u, v) = UV + 1$$

where U (resp., V) is a polynomial in u (resp., v) of degree at most 1, with coefficients in $\{0,1\}$. Given that fact, we rewrite any game (δ, f) using two expressions from the above, one for δ, and the other for f. We thus get nine different types of games, which can be narrowed down to five types by noticing that games like $A + B = XY + 1$ are the same as games like $A' + B' = X'Y'$, up to the (reversible) transformation $B' = B + 1$. These five types of games are the following:

$$\begin{aligned}
\delta(a,b) = A + B = f(x,y) = X + Y \\
\delta(a,b) = A + B = f(x,y) = XY \\
\delta(a,b) = AB = f(x,y) = X + Y \\
\delta(a,b) = AB = f(x,y) = XY \\
\delta(a,b) = AB = f(x,y) = XY + 1
\end{aligned}$$

Since f (resp., δ) depends on both x and y (resp., both a and b), all polynomials in these five types of games are of degree exactly 1, hence making all transformations reversible. Therefore, if two games can be rewritten into the same type, then they are equivalent. Table 1 describes the equivalence classes over the set of games formed by the five types above, and provides a representative for each class.

Table 1. Equivalence classes for true 2-player games depending on both inputs. The first two classes of games are deterministic, i.e., can be solved by a deterministic protocol. Instead, the last three classes are not deterministic. No deterministic protocol can solve any of the games in these three classes.

	Form of the class	Representative of the class
Deterministic	$AB = XY$	PROD $a \wedge b = x \wedge y$
	$A + B = X + Y$	SUM $a \oplus b = x \oplus y$
Not deterministic	$A + B = XY$	CHSH $a \oplus b = x \wedge y$
	$AB = X + Y$	NEQ $a \wedge b = x \oplus y$
	$AB = XY + 1$	AMOS $a \wedge b = \neg(x \wedge y)$

The theorem holds for games PROD and SUM since both of them can be solved by a deterministic protocol. Every game that is neither equivalent to an XOR-game nor deterministic is equivalent to an AND-game: NEQ or AMOS. We now show that non-local boxes fail to solve AMOS or NEQ with higher probabilistic guarantee than what can be achieved with local boxes.

Let us first examine AMOS. We start by showing that any box that solves AMOS with probabilistic guarantee $p > \frac{2}{3}$ is signaling. Suppose that there exists a non-signaling box B, defined by the correlation $\Pr(a, b|x, y)$, that solves AMOS with probability p. On the one hand, for any probability distribution $\pi = \{\pi_{xy} | (x, y) \in \{0, 1\}^2\}$ of the inputs, we have

$$\sum_{xy} \pi_{xy} \Pr(\text{success for input } (x, y)) \geq p$$

On the other hand, we have

$$\sum_{xy} \pi_{xy} \Pr(\text{success for input } (x, y)) = \sum_{xy} \pi_{xy} \sum_{ab} \mathbb{1}_{\{a \wedge b = \neg(x \wedge y)\}} \Pr(a, b|x, y)$$

where $\mathbb{1}_{\{a \wedge b = \neg(x \wedge y)\}}$ denotes the boolean indicator function of whether $a \wedge b = \neg(x \wedge y)$ is true or not. Let us consider the following distribution π^*:

$$\pi^*_{00} = 0 \quad \text{and} \quad \pi^*_{xy} = \frac{1}{3} \text{ for all } (x, y) \neq (0, 0)$$

Let $p_{abxy} = \Pr(a, b|x, y)$ for box B. The probability of success with the input distribution π^* satisfies

$$\begin{aligned}
\sum_{xy} \pi^*_{xy} \Pr(\text{success for } (x, y)) &= \frac{1}{3} \sum_{xy \neq (0,0)} \mathbb{1}_{\{a \wedge b = \neg(x \wedge y)\}} p_{abxy} \\
&= \frac{1}{3} \left(p_{1101} + p_{1110} + \sum_{(ab) \neq (11)} p_{ab11} \right) \\
&= \frac{1}{3} \left(p_{1101} + p_{1110} + p_{0011} + p_{0111} + p_{1011} \right) \quad (4)
\end{aligned}$$

The non-signaling conditions (cf., Eq. 3) require that, for every a, b, x, y,

$$\begin{aligned}
p_{a0x0} + p_{a1x0} &= p_{a0x1} + p_{a1x1} \\
\text{and } p_{0b0y} + p_{1b0y} &= p_{0b1y} + p_{1b1y}
\end{aligned}$$

which gives a bound on the first two terms of Equation 4:

$$\begin{aligned}
p_{1101} &= p_{1111} + p_{0111} - p_{0101} \leq p_{1111} + p_{0111} \\
\text{and } p_{1110} &= p_{1111} + p_{1011} - p_{1010} \leq p_{1111} + p_{1011}
\end{aligned}$$

The probability p of success is therefore bounded by :

$$\begin{aligned}
p &\leq \frac{1}{3} \left(p_{1111} + p_{0111} + p_{1011} + p_{1111} + p_{0011} + p_{0111} + p_{1011} \right) \\
&\leq \frac{1}{3} \Big((2 \sum_{ab} p_{ab11}) - p_{1111} \Big) \\
&\leq \frac{2}{3}
\end{aligned}$$

as $\sum_{ab} p_{abxy} = 1$ for any fixed (x, y), and $p_{1111} \geq 0$. Therefore, every non-signaling box solves AMOS with success at most $\frac{2}{3}$.

Regarding NEQ, we observe that with distribution π^*, AMOS and NEQ become the same games:

$$f_{\text{AMOS}}(x, y) = f_{\text{NEQ}}(x, y)$$

for all $(x, y \neq (0, 0)$. As a consequence, the same bound $\frac{2}{3}$ holds for NEQ: every non-signaling box solves NEQ with success at most $\frac{2}{3}$.

We now show that the bound $\frac{2}{3}$ for AMOS and NEQ can be reached by local boxes. For this purpose, we describe a protocol using solely shared randomness, and reaches success probability $\frac{2}{3}$. Let a_0 and a_1 (resp., b_0 and b_1) be the outputs of Alice (resp. Bob) on the respective input $x = 0$ and $x = 1$ (resp., $y = 0$ and $y = 1$). AMOS translates into solving the system:

$$\begin{cases} a_0 \cdot b_0 = 1 \\ a_0 \cdot b_1 = 1 \\ a_1 \cdot b_0 = 1 \\ a_1 \cdot b_1 = 0 \end{cases} \tag{5}$$

and NEQ translates into :

$$\begin{cases} a_0 \cdot b_0 = 0 \\ a_0 \cdot b_1 = 1 \\ a_1 \cdot b_0 = 1 \\ a_1 \cdot b_1 = 0 \end{cases} \tag{6}$$

The second and third equations of the system for AMOS as well as for NEQ imply that $a_0 = a_1 = b_0 = b_1 = 1$, resulting in the last equation impossible to be satisfied in both games. Hence the last three equations of each system cannot be simultaneously satisfied. Instead, if one chooses to ignore one of them, then one can find a solution to the game. Playing any one of the two games using shared randomness, we allow Alice and Bob to have access, before knowing their inputs, to a shared random variable λ uniformly distributed in $\{1, 2, 3\}$, designating the equation to be ignored among the last three ones. Alice and Bob will fail to solve the game with probability at most $\frac{1}{3}$ (when the ignored equation is precisely the one corresponding to the actual inputs), making the success probability for any input (x, y) equal to $\frac{2}{3}$. This completes the proof of the theorem. □

It turns out that even relaxing the constraints placed on solving the game, by considering average case analysis, does not allow non-signaling boxes to perform better than local boxes on games not equivalent to XOR-games.

Theorem 2. *Let (δ, f) be a 2-player game that is not equivalent to any* XOR*-game. Let p be the largest* average *success probability for (δ, f) over all local boxes. Then every box solving (δ, f) with average probabilistic guarantee $> p$ is signaling.*

Proof. Using the same arguments as in the proof of Theorem 1, we limit the analysis to AMOS and NEQ. For average case analysis, we consider these two

games with input probability distribution $\pi_{xy} = \frac{1}{4}$ for every $(x, y) \in \{0, 1\}^2$. The success probability for Alice and Bob with this input distribution is then given by:

$$\Pr(\text{success}) = \frac{1}{4} \sum_{x,y} \sum_{a,b} \mathbb{1}_{\{\delta(a,b)=f(x,y)\}} \Pr(a, b|x, y)$$

First, we show that the protocol described in the proof of Theorem 1 for solving AMOS and NEQ has average success probability $\frac{3}{4}$. Indeed, the success probability of that protocol can be written as:

$$\Pr(\text{success}) = \frac{1}{4} \sum_{x,y} \Pr(\text{success}(x, y)) = \frac{1}{4}\Big(1 + \frac{2}{3} + \frac{2}{3} + \frac{2}{3}\Big) = \frac{3}{4}$$

because, the protocol always satisfies the first equation of both games, and satisfies each of the three other equations (of both games) with probability $\frac{2}{3}$.

Next, we show that a non-local box cannot solve AMOS or NEQ with average success probability greater than $\frac{3}{4}$. Indeed, we have

$$\Pr(\text{success}) = \frac{1}{4}\Big[\Big(\sum_{(x,y)\neq(0,0)} \sum_{a,b} \mathbb{1}_{\{\delta(a,b)=f(x,y)\}} \Pr(a, b|x, y)\Big) + \Big(\sum_{a,b} \mathbb{1}_{\{\delta(a,b)=f(0,0)\}} \Pr(a, b|0, 0)\Big)\Big]$$

The first term is the same as the one analyzed in the proof of Theorem 1, where it was proved to be at most 2. The second term is at most $\sum_{ab} \Pr(a, b|0, 0) \leq 1$. Therefore, the average success probability for non-local boxes is at most $\frac{3}{4}$. □

The practical interest of the previous two theorems comes from their consequence to distributed quantum computing:

Corollary 1. *Quantum correlations does not help for solving 2-player games that are not equivalent to any* XOR*-game. This limitation holds for both worst case, and average case analysis.*

4 Open Problem

One obvious generalization of the 2-player games is to consider games with more than two players, with IDs from 1 to $n \geq 2$. In the n-player game (δ, f), Player i receives boolean x_i as input, and must return a boolean a_i such that

$$\delta(a_1, \ldots, a_n) = f(x_1, \ldots, x_n)$$

in absence of communication between the players. As for two players, two classes of games deserve specific interest:

- XOR-games: $\delta(a_1, \ldots, a_n) = a_1 \oplus \ldots \oplus a_n$, for they generalize the CHSH game, and for they can be solved by a non-signaling box implementable by a circuit of PR boxes (see [2]);

– AND-games: $\delta(a_1, \ldots, a_n) = a_1 \wedge \ldots \wedge a_n$ for they correspond to the standard decision mechanism in the distributed computing literature (see, e.g., [14]).

In particular, the n-player variant of AMOS is:

$$\bigwedge_{i=1}^{n} a_i = \bigwedge_{i \neq j} (\overline{x_i \wedge x_j}).$$

There exists a randomized protocol (see [10]), that is using individual random coins, and solves AMOS with success guarantee $\frac{\sqrt{5}-1}{2} \geq 0.61 > 1/2$. In this protocol, every selected player (i.e., one with input 1) outputs 1 with probability p, to be fixed later, and 0 with probability $1-p$. Every non-selected player (i.e., one with input 0) systematically outputs 0. Hence, if no players are selected, then the protocol always outputs the right answer. If one player is selected, then the protocol fails with probability $1-p$, while if two or more players are selected then the protocol fails with probability at most p^2. Solving $p^2 = 1-p$ results in picking the optimal probability $p^* = \frac{\sqrt{5}-1}{2}$.

On the other hand, we have seen in this paper that AMOS can be solved with success guarantee $\frac{2}{3} > p^*$ by two players applying a probabilistic protocol using shared randomness. One can actually show that the same guarantee can be achieved with three players, by analyzing the following system

$$\begin{cases} a_0 \cdot b_0 \cdot c_0 = 1 \\ a_1 \cdot b_0 \cdot c_0 = 1 \\ a_0 \cdot b_1 \cdot c_0 = 1 \\ a_0 \cdot b_0 \cdot b_1 = 1 \\ a_1 \cdot b_1 \cdot c_0 = 0 \\ a_1 \cdot b_0 \cdot c_1 = 0 \\ a_0 \cdot b_1 \cdot c_1 = 0 \\ a_1 \cdot b_1 \cdot b_1 = 0 \end{cases}$$

which lists the eight equations for AMOS corresponding to the eight possible inputs of the games. Consider the protocol which solves that system after ignoring the second and seventh equations with probability $\frac{1}{3}$, the third and sixth with probability $\frac{1}{3}$, and the fourth and fifth with probability $\frac{1}{3}$. This protocol has success probability at least $\frac{2}{3}$ for every triple of inputs.

Unfortunately, the protocols for two and three players do not seem to extend easily to a higher number of players. For four players, we have designed an ad hoc probabilistic protocol using shared randomness, with success probability $\frac{9}{14} > \frac{\sqrt{5}-1}{2}$, but we failed to design a local protocol with success probability $\frac{2}{3}$. For more than four players, the ad hoc protocol could be generalized, but we have not identified a general pattern for it.

Instead, the lower bound $\frac{2}{3}$ on the probability of success for solving AMOS with non-signaling boxes established in this paper trivially extends to n players. We thus conclude by stating the following problem.

Open problem: Prove or disprove the existence of a shared-randomness probabilistic protocol that solves the n-player AMOS game with success probability $\frac{2}{3}$, for all $n \geq 2$.

References

1. Barrett, J., Linden, N., Massar, S., Pironio, S., Popescu, S., Roberts, D.: Nonlocal correlations as an information-theoretic resource. Physical Review A 71(2), 1–11 (2005)
2. Barrett, J., Pironio, S.: Popescu-Rohrlich correlations as a unit of nonlocality. Phys. Rev. Lett. 95(14) (2005)
3. Bell, J.S.: On the Einstein-Podolsky-Rosen paradox. Physics 1(3), 195–200 (1964)
4. Buhrman, H., Cleve, R., Massar, S., de Wolf, R.: Non-locality and communication complexity. Reviews of Modern Physics 82, 665–698 (2010)
5. Cirel'son, B.S.: Quantum generalizations of bell's inequality. Letters in Math. Phys. 4(2), 93–100 (1980)
6. Clauser, J.F., Horne, M.A., Shimony, A., Holt, R.A.: Proposed experiment to test local hidden-variable theories. Physical Review Letters 23(15), 880–884 (1969)
7. Das Sarma, A., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., Wattenhofer, R.: Distributed verification and hardness of distributed approximation. In: 43rd ACM Symp. on Theory of Computing, STOC (2011)
8. Dupuis, F., Gisin, N., Hasidim, A., Allan Méthot, A., Pilpel, H.: No nonlocal box is universal. J. Math. Phys. 48(082107) (2007)
9. Einstein, A., Podolsky, B., Rosen, N.: Can quantum-mechanical description of physical reality be considered complete? Physical Review 47(10), 777–780 (1935)
10. Fraigniaud, P., Korman, A., Peleg, D.: Local distributed decision. In: 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 708–717 (2011)
11. Fraigniaud, P., Rajsbaum, S., Travers, C.: Locality and Checkability in Wait-Free Computing. In: Peleg, D. (ed.) DISC 2011. LNCS, vol. 6950, pp. 333–347. Springer, Heidelberg (2011)
12. Fraigniaud, P., Rajsbaum, S., Travers, C.: Universal distributed checkers and orientation-detection tasks (submitted, 2012)
13. Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. Distributed Computing 22, 215–233 (2010)
14. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM J. Comput. 24(6), 1259–1277 (1995)
15. Popescu, S., Rohrlich, D.: Quantum nonlocality as an axiom. Foundations of Physics 24(3), 379–385 (1994)

On Bidimensional Congestion Games*

Vittorio Bilò[1], Michele Flammini[2], and Vasco Gallotti[2]

[1] Dipartimento di Matematica e Fisica "Ennio De Giorgi", Università del Salento
Provinciale Lecce-Arnesano, P.O. Box 193, 73100 Lecce, Italy
vittorio.bilo@unisalento.it
[2] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica,
Università di L'Aquila
Via Vetoio, Loc. Coppito, 67100 L'Aquila, Italy
{michele.flammini,vasco.gallotti}@univaq.it

Abstract. We introduce multidimensional congestion games, that is, congestion games whose set of players can be partitioned into $k+1$ clusters $C_0, C_1, \ldots, C_k$. Players in C_0 have full information about all the other participants in the game, while players in C_i, for any $1 \leq i \leq k$, have full information only about the members of $C_0 \cup C_i$ and are unaware of all the other ones. This model has at least two interesting applications: (i) it is a special case of graphical congestion games in which the game's social knowledge graph is undirected and has independence number equal to k, and (ii) it models scenarios in which players may be of different types and the level of competition that each player experiences on a resource depends on the player's type and on the types of the other players sharing the resource. We focus on the case in which $k = 2$ and the cost function associated with each resource is linear and show bounds on the prices of anarchy and stability for two different social functions.

1 Introduction

Congestion games are, perhaps, the most famous class of non-cooperative games due to their capability to model several interesting competitive scenarios, while maintaining some nice properties. In these games there is a set of players sharing a set of resources. Each resource has an associate cost function which depends on the number of players using it (the so-called *congestion*). Players aim at choosing subsets of resources so as to minimize the sum of their costs. *Weighted congestion games* is the generalization in which each player has a weight and the congestion of a resource becomes the sum of the weights of its users (thus, congestion games correspond to weighted congestion games in which all players have unitary weight).

* This work was partially supported by the PRIN 2008 research project COGENT "Computational and game-theoretic aspects of uncoordinated networks" funded by the Italian Ministry of University and Research.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 147–158, 2012.

Congestion games have been introduced by Rosenthal in [16]. He proved that each such a game admits a bounded potential function whose set of local minima coincides with the set of *pure Nash equilibria* of the game, that is, strategy profiles in which no player can decrease her cost by unilaterally changing her strategic choice. This existence result makes congestion games particularly appealing especially in all those applications in which pure Nash equilibria are elected as the ideal solution concept.

In these contexts, the study of the inefficiency due to selfish and non-cooperative behavior has affirmed as a fervent research direction. To this aim, the notions of *price of anarchy* (Koutsoupias and Papadimitriou [13]) and *price of stability* (Anshelevich *et al.* [2]) are widely adopted. The price of anarchy (resp. stability) compares the performance of the worst (resp. best) pure Nash equilibrium with that of an optimal cooperative solution.

Congestion games with unrestricted cost functions are general enough to model the Prisoner's Dilemma game, whose unique pure Nash equilibrium is known to perform arbitrarily bad with respect to the solution in which all players cooperate. Hence, in order to deal with significative bounds on the prices of anarchy and stability, some kind of regularity needs to be imposed on the cost functions associated with the resources. To this aim, lot of research attention has been devoted to the case of polynomial cost functions [1,3,4,5,7,8,9,10].

Among these, the particular case of linear functions has been successfully characterized: Christodoulou and Koutsoupias [8] showed that the price of anarchy is $5/2$, while the works of Caragiannis *et al.* [7] and Christodoulou and Koutsoupias [9] set the price of stability to $1 + 1/\sqrt{3}$. Moreover, it has been shown in [12,14,15] that this is the only case, together with that (perhaps not particularly meaningful) of exponential cost functions, for which even weighted congestion games admit a potential function. For these games, Awerbuch *et al.* [3] gave a price of anarchy of $(3 + \sqrt{5})/2$.

Motivations and Previous Works. Traditional congestion games are defined under a *full information* scenario: each player knows all the other participants in the game as well as their available strategies. These requirements, anyway, become too constraining in many practical applications, where players may be unaware about even the mere existence of other potential competitors. This observation, together with the widespread of competitive applications in social networks, has drawn some attention on the model of *graphical congestion games*.

A graphical congestion game $(\mathcal{G}, G)$ is obtained by coupling a traditional congestion game $\mathcal{G}$ with a *social knowledge graph* G expressing the *social context* in which the players operate. In G, each node is associated with a player of $\mathcal{G}$ and there exists a directed edge from node i to node j if and only if player i has full information about player j. A basic property of congestion games is that the congestion of a resource r in a given strategy profile S, that is the number of players choosing r in S, is the same for all players. The existence of a social context in graphical congestion games, instead, makes the congestion of each resource player dependent. In these games, in fact, the congestion presumed by

player i on resource r in the strategy profile S is obtained by excluding from the set of players choosing r in S those of whom player i has no knowledge. Clearly, if G is complete, then there is no difference between $(\mathcal{G}, G)$ and $\mathcal{G}$. In all the other cases, however, there may be a big difference between the cost that a player *presumes* to pay on a certain strategy profile and the real cost that she effectively *perceives* because of the presence of players she was unaware of during her decisional process.

Graphical congestion games have been introduced by Bilò *et al.* in [6]. They focus on linear cost functions and provide a complete characterization of the cases in which existence of pure Nash equilibria can be guaranteed. They show that equilibria always exist if and only if G is either undirected or directed acyclic. Then, for all these cases, they give bounds on the price of anarchy and stability expressed as a function of the number of players in the game and the maximum degree of G when the social function measuring the overall quality of a profile is either the sum of the perceived costs or the sum of the presumed ones.

Fotakis *et al.* [11] argue that the maximum degree of G is not a proper measure of the level of social ignorance in a graphical congestion game and propose to bound the prices of anarchy and stability as functions of the independence number of G, denoted, as usual, by $\alpha(G)$. They focus on games with weighted players and linear cost functions and show that they still admit a potential function when G is undirected. Then, they prove that, for these games, the price of anarchy is between $\alpha(G)(\alpha(G)+1)$ and $\alpha(G)(\alpha(G)+2+\sqrt{\alpha(G)^2+4\alpha(G)})/2$ when the social function is both the sum of the perceived costs and the sum of the presumed ones and that the price of stability is between $\alpha(G)$ and $2\alpha(G)$ when the social function is the sum of the perceived costs.

Our Contribution and Significance. The works of Bilò *et al.* [6] and Fotakis *et al.* [11] aim at characterizing the impact of social ignorance in the most general case, that is, without imposing any particular structure on the social knowledge graphs defining the games. Nevertheless, real world based knowledge relationships usually obey some regularities and present recurrent patterns: for instance, people tend to cluster themselves into well-structured collaborative groups (cliques) due to family memberships, mutual friendships, interest sharing, business partnerships.

To this aim, we introduce and study *multidimensional congestion games*, that is, congestion games whose set of players can be partitioned into $k+1$ clusters $C_0, C_1, \ldots, C_k$. Players in C_0 have full information about all the other participants in the game, while players in C_i, for any $1 \leq i \leq k$, have full information only about the members of $C_0 \cup C_i$ and are unaware of all the other ones. It is not difficult to see (and we provide a formal proof of this fact in the next section) that each multidimensional congestion game is a graphical congestion game whose social knowledge graph G is undirected and verifies $\alpha(G) = k$. In addition, G possesses the following, well-structured, topology: it is the union of

$k+1$ disjoint cliques (each corresponding to one of the $k+1$ clusters in the multidimensional congestion game) with the additional property that there exists an edge from all the nodes belonging to one of these cliques (the one corresponding to cluster C_0) to all the nodes in all the other cliques.

By these observations, we believe that the study of multidimensional congestion games, that is, graphical congestion games restricted to social knowledge graphs like these, may be better suited to understand the impact of social ignorance in non-cooperative systems coming from practical and real-world applications.

Moreover, the particular social knowledge relationships embedded in the definition of multidimensional congestion games, perfectly model the situation that generates when several independent games with full information are gathered together by some promoting parties so as to form a sort of "global super-game". The promoting parties become players with full information in the super-game, while each player in the composing sub-games maintains full information about all the other players in the same sub-game, acquires full information about all the promoting parties in the super game, but completely ignores the players in the other sub-games. Such a composing scheme resembles, in a sense, the general architecture of the Internet, viewed as a self-emerged network resulting from the aggregation of several autonomous systems (AS). Users in an AS have full information about anything happening within the AS, but, at the same time, they completely ignore the networks's global architecture and how it develops outside their own AS, except for the existence of high-level network routers. High-level network routers, instead, have full information about the entire network.

Furthermore, multidimensional congestion games are also useful to model games in which players may belong to different types and the level of competition that each player experiences on each selected resource depends on her type and on the types of the other players sharing the resource. Consider, for instance, a machine which is able to perform k different types of activities in parallel and a set of tasks requiring the use of such a machine. Tasks are of two types: simple and complex. Simple tasks take the machine busy on one particular activity only, while complex tasks require the completion of all the k activities. Hence, complex tasks compete with all the other tasks, while simple ones compete only with the tasks requiring the same machine (thus, also with complex tasks). A similar example is represented by a facility location game where players want to locate their facilities so as to minimize the effect of the competition due to the presence of neighbor competitors. If we assume that the facilities can be either specialized shops selling particular products (such as perfumeries, clothes shops, shoe shops) or shopping centers selling all kinds of products, we have again that the shopping centers compete with all the other participants in the game, while specialized shops compete only with shops of the same type and with shopping centers.

In this paper, we focus on bidimensional congestion games, that is, the case in which $k = 2$. We also assume that the cost function associated with each

resource is linear and the players are unweighted. In such a setting, we bound the prices of anarchy and stability with respect to the two social functions sum of the perceived costs and sum of the presumed costs. In fact, when multidimensional congestion games are viewed as graphical congestion games with highly clustered knowledge relationships, the social function sum of the perceived costs is the more appropriate to define the overall quality of a profile: players decide according to their knowledge, but then, when the solution is physically realized, their cost becomes influenced also by the players of which they were not aware. Hence, under this social function, the notions of prices of anarchy and stability effectively measure the impact of social ignorance in these kind of games. On the other hand, when multidimensional congestion games are used to model players belonging to different types, the perceived cost of a player coincides with the presumed one since there is no real social ignorance, even if the fact that players can be of different types allows for a reinterpretation of the model as a special case of graphical congestion games. Hence, in such a setting, the inefficiency due to selfish behavior has to be analyzed with respect to the social function sum of the presumed costs. We show that the price of anarchy is $119/33$ for the social function sum of the presumed costs and it is $35/8$ for the social function sum of the perceived ones, and that the price of stability is between $(1+\sqrt{5})/2 \approx 1.618$ and $1+2/\sqrt{7} \approx 1.756$ for the social function sum of the presumed costs and between $(5+\sqrt{17})/4 \approx 2.28$ and 2.92 for the social function sum of the perceived ones. Our results are derived by exploiting the primal-dual method recently developed in [5].

Paper Organization. Next section contains all formal definitions, notation, and some numerical lemmas. In Sections 3 and 4, we present our bounds for the price of anarchy and the price of stability, respectively. Due to space limitations, some proofs have been omitted and will be given in the full version of the paper.

2 Model, Definitions and Numerical Lemmas

In a congestion game $\mathcal{G}$, we have n players and a set of resources R, where each resource $r \in R$ has an associated cost function ℓ_r. The set of strategies for each player $i \in [n]$, denoted as S_i, can be any subset of the powerset of R, that is, $S_i \subseteq 2^R$. Given a strategy profile $S = (s_1, \ldots, s_n)$, the congestion of resource r in the profile S, denoted as $c_r(S)$, is the number of players choosing r in S, that is, $c_r(S) = |\{i \in [n] : r \in s_i\}|$. The cost paid by player i in S is $\omega_i(S) = \sum_{r \in s_i} \ell_r(c_r(S))$. A *linear congestion game* is a congestion game such that, for any $r \in R$, it holds $\ell_r(x) = \alpha_r x + \beta_r$, with $\alpha_r, \beta_r \geq 0$.

Given a strategy profile S and a strategy $s' \in S_i$ for a player $i \in [n]$, we denote with $S_{-i} \diamond s'$ the strategy profile obtained from S by replacing the strategy played by i in S with s'. A *pure Nash equilibrium* is a strategy profile S such that, for any player $i \in [n]$ and for any strategy $s' \in S_i$, it holds $\omega_i(S_{-i} \diamond s) \geq \omega_i(S)$.

A k-dimensional congestion game $(\mathcal{G},\mathcal{C})$ consists of a congestion game $\mathcal{G}$ whose set of players is partitioned into $k+1$ clusters $C_0, C_1, \ldots, C_k$. We say that players in C_0 are *omniscient* and that players in C_i, for any $1 \leq i \leq k$, are *ignorant*. Given a strategy profile $S = (s_1, \ldots, s_n)$, we denote with $c_i^r(S)$ the congestion presumed by player i on resource r in the profile S. For an ignorant player $i \in C_j$, we have $c_i^r(S) = |\{u \in C_0 \cup C_j : r \in s_u\}|$, while for an omniscient player i, we have $c_i^r(S) = |\{j \in [n] : r \in s_j\}|$. The cost of player i in S is $\omega_i(S) = \sum_{r \in s_i} \ell_r(c_i^r(S))$. A k-dimensional linear congestion game is a pair $(\mathcal{G},\mathcal{C})$ such that $\mathcal{G}$ is a linear congestion game.

A graphical congestion game $(\mathcal{G}, G)$ consists of a congestion game $\mathcal{G}$ and a directed graph $G = (N, A)$ such that each node of N is associated with a player in $\mathcal{G}$ and there exists a directed edge from node i to node j if and only if player i has full information about player j. Given a strategy profile $S = (s_1, \ldots, s_n)$, let $G_r(S) = (N_r, A_r)$ be the subgraph of G induced by the set of players who are selecting r in S. We denote with $n_r(S) = |N_r(S)|$ the number of nodes in $G_r(S)$, with $m_r(S) = |A_r(S)|$ the number of edges in $G_r(S)$ and with $\delta_i^r(S)$ the out-degree of node i in $G_r(S)$. The congestion presumed by player i on resource r in the profile S is $c_i^r(S) = 1 + \delta_i^r(S)$ and the cost paid by player i in S is $\omega_i(S) = \sum_{r \in s_i} \ell_r(c_i^r(S))$. A graphical linear congestion game is a pair $(\mathcal{G}, G)$ such that $\mathcal{G}$ is a linear congestion game.

A function $\Phi : \mathcal{S} \mapsto \mathbb{R}$ is an *exact potential function* for a graphical congestion game $(\mathcal{G}, G)$, if for any profile S, any player $i \in [n]$ and any strategy $s' \in S_i$, it holds $\Phi(S) - \Phi(S_{-i} \diamond s') = \omega_i(S) - \omega_i(S_{-i} \diamond s')$. In [6], it is shown that each graphical linear congestion game $(\mathcal{G}, G)$ such that G is undirected admits the exact potential function $\Phi(S) = \sum_{r \in R}(\alpha_r m_r(S) + (\alpha_r + \beta_r) n_r(S))$.

The following result shows that k-dimensional congestion games are instances of graphical congestion games.

Proposition 1. *Each k-dimensional congestion game is a graphical congestion game whose social knowledge graph is undirected.*

Each game admitting an exact potential function always admits pure Nash equilibria. Hence, by Proposition 1 and the existence of an exact potential for graphical linear congestion games with undirected social knowledge graphs, we have that k-dimensional linear congestion games always admit pure Nash equilibria.

Let $\mathcal{S}$ be the set of all possible strategy profiles which can be realized in $(\mathcal{G},\mathcal{C})$, we denote with $\mathcal{NE}(\mathcal{G},\mathcal{C}) \subseteq \mathcal{S}$ the set of pure Nash equilibria of $\mathcal{G}$. Let $\mathsf{SF} : \mathcal{S} \mapsto \mathbb{R}_{\geq 0}$ be a *social function* measuring the overall quality of each strategy profile in $\mathcal{S}$. We denote with S^* the *social optimum* of $(\mathcal{G},\mathcal{C})$ with respect to SF, that is, the strategy profile minimizing the social function SF.

The *price of anarchy* of game $(\mathcal{G},\mathcal{C})$, denoted by $PoA(\mathcal{G},\mathcal{C})$, with respect to SF is the ratio between the social value of the *worst* Nash equilibrium of $(\mathcal{G},\mathcal{C})$ and that of the social optimum, i.e., $PoA(\mathcal{G},\mathcal{C}) = \max_{S \in \mathcal{NE}(\mathcal{G},\mathcal{C})} \frac{\mathsf{SF}(S)}{\mathsf{SF}(S^*)}$. The *price of stability* of game $(\mathcal{G},\mathcal{C})$, denoted by $PoS(\mathcal{G},\mathcal{C})$, with respect to SF is the ratio between the social value of the *best* Nash equilibrium of $(\mathcal{G},\mathcal{C})$ and that of the social optimum, i.e., $PoS(\mathcal{G},\mathcal{C}) = \min_{S \in \mathcal{NE}(\mathcal{G},\mathcal{C})} \frac{\mathsf{SF}(S)}{\mathsf{SF}(S^*)}$.

In this paper, we focus on bidimensional linear congestion games for which we add some further notation. Fix a strategy profile $S = (s_1, \ldots, s_n)$. For any resource $r \in R$, let $S_r = |\{i \in [n] : r \in s_i\}|$ be the number of players using r in S, $S'_r = |\{i \in C_1 : r \in s_i\}|$ be the number of players belonging to C_1 using r in S and $S''_r = |\{i \in C_2 : r \in s_i\}|$ be the number of players belonging to C_2 using r in S. As already showed in [5] for the case of linear congestion games, we do not lose in generality by restricting the analysis of both the prices of anarchy and stability to games in which the latency functions are of the form $\ell_r(x) = \alpha_r x$ as long as the social function SF is defined as a function of the players' costs in a given strategy profile. Hence, the potential given in [6] for graphical linear congestion games becomes $\Phi(S) = \sum_{r \in R}(\alpha_r(S_r^2 + S_r - 2S'_rS''_r))$, the social function sum of presumed latencies becomes $\textsc{Sum}_{pres}(S) = \sum_{i \in [n]} \omega_i(S) = \sum_{r \in R}(\alpha_r(S_r^2 - 2S'_rS''_r))$ while the social function sum of perceived latencies becomes $\textsc{Sum}_{perc}(S) = \sum_{r \in R}(\alpha_r S_r^2)$. For a fixed bidimensional game $(\mathcal{G}, \mathcal{C})$, we will denote with $E = (e_1, \ldots, e_n)$ a pure Nash equilibrium of $(\mathcal{G}, \mathcal{C})$ and with $O = (o_1, \ldots, o_n)$ the social optimum for $(\mathcal{G}, \mathcal{C})$ under some social function SF.

We conclude the section by providing all the technical lemmas needed to prove our main theorems.

Lemma 1. *Let $\theta : \mathbb{Z}^6_{\geq 0} \mapsto \mathbb{Q}$ be the function such that $\theta(a, b, c, d, e, f) = 18a^2 - a(b + c + 51d - e - f) + 50bf + 50ce - 34bc + 119d^2 - 51d + e + f - 238ef$. For any $(a, b, c, d, e, f) \in \mathbb{Z}^6_{\geq 0}$ such that $a \geq b + c$ and $d \geq e + f$, it holds $\theta(a, b, c, d, e, f) \geq 0$.*

Lemma 2. *Let $\theta : \mathbb{Z}^6_{\geq 0} \mapsto \mathbb{Q}$ be the function such that $\theta(a, b, c, d, e, f) = 7a^2 + 3a(2b + 2c - 5d - 2e - 2f) + 21bf + 21ce - 42bc + 35d^2 - 15d - 6e - 6f$. For any $(a, b, c, d, e, f) \in \mathbb{Z}^6_{\geq 0}$ such that $a \geq b+c$ and $d \geq e+f$, it holds $\theta(a, b, c, d, e, f) \geq 0$.*

Lemma 3. *Let $\theta : \mathbb{Z}^6_{\geq 0} \mapsto \mathbb{Q}$ be the function such that $\theta(a, b, c, d, e, f) = a^2(3 - \sqrt{7}) - a(2d - 1 - \sqrt{7}) + 2bc(\sqrt{7} - 3) + 2(bf + ce) + (d^2 - d)(3 + \sqrt{7}) - 2(3 + \sqrt{7})ef$. For any $(a, b, c, d, e, f) \in \mathbb{Z}^6_{\geq 0}$ such that $a \geq b + c$ and $d \geq e + f$, it holds $\theta(a, b, c, d, e, f) \geq 0$.*

Lemma 4. *Let $\theta : \mathbb{Z}^6_{\geq 0} \mapsto \mathbb{Q}$ be the function such that $\theta(a, b, c, d, e, f) = 49a^2 + a(62b + 62c - 68d - 62e - 62f + 81) + 130bf + 130ce - 422bc + 211d^2 - 149d + 162ef - 62e - 62f$. For any $(a, b, c, d, e, f) \in \mathbb{Z}^6_{\geq 0}$ such that $a \geq b + c$ and $d \geq e + f$, it holds $\theta(a, b, c, d, e, f) \geq 0$.*

3 Bounds for the Price of Anarchy

We apply the primal-dual technique introduced in [5]. For any fixed pair of profiles E and O, consider the problem of maximizing the ratio $\frac{\textsc{Sum}_{pres}(E)}{\textsc{Sum}_{pres}(O)}$. This yields the following primal linear program.

$$
\begin{aligned}
&\textit{maximize} \sum_{r\in R} \left(\alpha_r(E_r^2 - 2E'_rE''_r)\right)\\
&\textit{subject to}\\
&\sum_{r\in e_i} (\alpha_r E_r) - \sum_{r\in o_i} (\alpha_r(E_r+1)) \le 0, \forall i \in C_0\\
&\sum_{r\in e_i} (\alpha_r(E_r - E''_r)) - \sum_{r\in o_i} (\alpha_r(E_r+1-E''_r)) \le 0, \forall i \in C_1\\
&\sum_{r\in e_i} (\alpha_r(E_r - E'_r)) - \sum_{r\in o_i} (\alpha_r(E_r+1-E'_r)) \le 0, \forall i \in C_2\\
&\sum_{r\in R} \left(\alpha_r(O_r^2 - 2O'_rO''_r)\right) = 1,\\
&\alpha_r \ge 0,\ \forall r \in R
\end{aligned}
$$

The two strategy profiles E and O play the role of fixed constants, while the multiplicative coefficients α_r in the cost functions associated with each resource $r \in R$ are variables that must be suitably chosen so as to satisfy certain desiderata. In particular, the first three constraints assure that in the strategy profile E no player can lower her cost by deviating to the strategy she plays in the optimal profile O (i.e., E is a pure Nash equilibrium), while the fourth constraint simply normalizes to 1 the value of the social optimum $\textsc{Sum}_{pres}(O)$. The objective function aims at maximizing the social value $\textsc{Sum}_{pres}(E)$ which, being the social optimum normalized to 1, is equivalent to maximize the ratio $\frac{\textsc{Sum}_{pres}(E)}{\textsc{Sum}_{pres}(O)}$. The dual formulation is

$$
\begin{aligned}
&\textit{minimize } \gamma\\
&\textit{subject to}\\
&\sum_{i\in C_0: r\in e_i} (x_i E_r) + \sum_{i\in C_1: r\in e_i} (y_i(E_r - E''_r))\\
&+ \sum_{i\in C_2: r\in e_i} (z_i(E_r - E'_r)) - \sum_{i\in C_0: r\in o_i} (x_i(E_r+1))\\
&- \sum_{i\in C_1: r\in o_i} (y_i(E_r+1-E''_r)) - \sum_{i\in C_2: r\in o_i} (z_i(E_r+1-E'_r))\\
&+\gamma(O_r^2 - 2O'_rO''_r) - E_r^2 + 2E'_rE''_r \ge 0,\ \forall r \in R\\
&x_i \ge 0, \forall i \in C_0\\
&y_i \ge 0, \forall i \in C_1\\
&z_i \ge 0, \forall i \in C_2
\end{aligned}
$$

By the Weak Duality Theorem, each feasible dual solution provides an upper bound on the optimal solution of the relative primal problem. Hence, by providing a feasible dual solution, we obtain an upper bound on the ratio $\frac{\textsc{Sum}_{pres}(E)}{\textsc{Sum}_{pres}(O)}$. Anyway, if the provided dual solution is independent on the particular choice of E and O, we obtain an upper bound on the ratio $\frac{\textsc{Sum}_{pres}(E)}{\textsc{Sum}_{pres}(O)}$ for any possible pair of profiles E and O, which means that we obtain an upper bound on the price of anarchy for the social function $\textsc{Sum}_{pres}$. For the case of the social function $\textsc{Sum}_{perc}$, we only need to replace the objective function and the fourth

constraint in the primal formulation with $\sum_{r\in R}\left(\alpha_r E_r^2\right)$ and $\sum_{r\in R}\left(\alpha_r O_r^2\right)=1$, respectively. This results in the deletion of the two terms $-2\gamma O'_r O''_r$ and $2E'_r E''_r$ occurring at the end of the dual constraints.

Theorem 1. *For each bidimensional linear congestion game $(\mathcal{G},\mathcal{C})$, it holds $PoA(\mathcal{G},\mathcal{C}) \leq \frac{119}{33}$ under the social function SUM_{pres} and $PoA(\mathcal{G},\mathcal{C}) \leq \frac{35}{8}$ under the social function SUM_{perc}.*

Proof. For the social function SUM_{pres}, set $\gamma=\frac{119}{33}$, $x_i=\frac{17}{11}$ for any $i\in C_0$, $y_i=\frac{50}{33}$ for any $i\in C_1$ and $z_i=\frac{50}{33}$ for any $i\in C_2$. With these values, for any $r\in R$, the dual constraint becomes

$$\frac{17}{11}E_r(E_r-E'_r-E''_r)+\frac{50}{33}(E_r(E'_r+E''_r)-2E'_rE''_r)-\frac{17}{11}(E_r+1)(O_r-O'_r-O''_r)$$

$$-\frac{50}{33}((E_r+1)(O'_r+O''_r)-E'_rO''_r-E''_rO'_r)+\frac{119}{33}(O_r^2-2O'_rO''_r)-E_r^2+2E'_rE''_r\geq 0$$

which is equivalent to

$$18E_r^2-E_r(E'_r+E''_r+51O_r-O'_r-O''_r)+50E'_rO''_r+50E''_rO'_r-34E'_rE''_r$$

$$+119O_r^2-51O_r+O'_r+O''_r-238O'_rO''_r\geq 0.$$

The claim follows by applying Lemma 1.

For the social function SUM_{perc}, set $\gamma=\frac{35}{8}$, $x_i=\frac{15}{8}$ for any $i\in C_0$, $y_i=\frac{21}{8}$ for any $i\in C_1$ and $z_i=\frac{21}{8}$ for any $i\in C_2$. With these values, for any $r\in R$, the dual constraint becomes

$$\frac{15}{8}E_r(E_r-E'_r-E''_r)+\frac{21}{8}(E_r(E'_r+E''_r)-2E'_rE''_r)-\frac{15}{8}(E_r+1)(O_r-O'_r-O''_r)$$

$$-\frac{21}{8}((E_r+1)(O'_r+O''_r)-E'_rO''_r-E''_rO'_r)+\frac{35}{8}O_r^2-E_r^2\geq 0$$

which is equivalent to

$$7E_r^2+3E_r(2E'_r+2E''_r-5O_r-2O'_r-2O''_r)+21E'_rO''_r+21E''_rO'_r-42E'_rE''_r$$

$$+35O_r^2-15O_r-6O'_r-6O''_r\geq 0.$$

The claim follows by applying Lemma 2. □

We now show the existence of two matching lower bounding instances.

Theorem 2. *There exist two bidimensional linear congestion games $(\mathcal{G},\mathcal{C})$ and $(\mathcal{G}',\mathcal{C}')$ such that $PoA(\mathcal{G},\mathcal{C})\geq\frac{119}{33}$ under the social function SUM_{pres} and $PoA(\mathcal{G}',\mathcal{C}')\geq\frac{35}{8}$ under the social function SUM_{perc}.*

Proof. For the social function SUM_{pres}, the lower bound is provided by the game depicted in Figure 1a), while for the social function SUM_{perc}, the lower bound is provided by the game depicted in Figure 1b). □

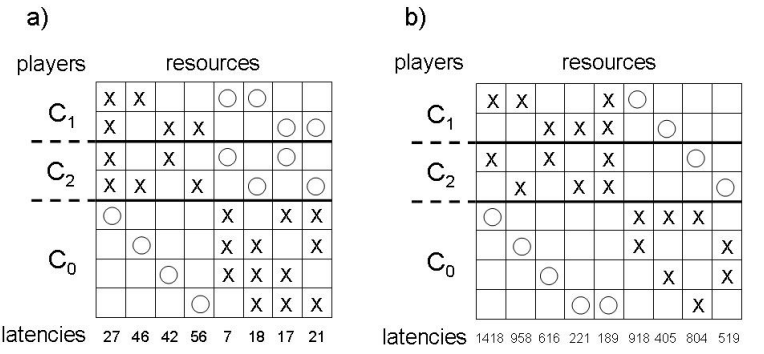


Fig. 1. Two games in which each player has exactly two strategies. Each column in the matrix represents a resource of cost function $\ell(x) = \alpha x$ whose coefficient α is reported at the bottom of the column. Each row i in the matrix models the strategy set of player i as follows: the circles represent all the resources belonging to o_i, while the crosses represent all the resources belonging to e_i.

4 Bounds for the Price of Stability

In order to bound the price of stability, we can use the same primal formulations exploited for the determination of the price of anarchy with the additional constraint $\Phi(E) \leq \Phi(O)$, which becomes $\sum_{r\in R}(E_r^2 + E_r - 2E'_rE''_r) \leq \sum_{r\in R}(O_r^2 + O_r - 2O'_rO''_r)$. Hence, the dual program for the social function SUM_{pres} becomes the following one.

$$
\begin{aligned}
&\textit{minimize } \gamma \\
&\textit{subject to} \\
&\sum_{i\in C_0: r\in e_i}(x_i E_r) + \sum_{i\in C_1: r\in e_i}(y_i(E_r - E''_r)) \\
&+ \sum_{i\in C_2: r\in e_i}(z_i(E_r - E'_r)) - \sum_{i\in C_0: r\in o_i}(x_i(E_r + 1)) \\
&- \sum_{i\in C_1: r\in o_i}(y_i(E_r + 1 - E''_r)) - \sum_{i\in C_2: r\in o_i}(z_i(E_r + 1 - E'_r)) \\
&+t(E_r^2 + E_r - 2E'_rE''_r - O_r^2 - O_r + 2O'_rO''_r) \\
&+\gamma(O_r^2 - 2O'_rO''_r) - E_r^2 + 2E'_rE''_r \geq 0, \ \forall r \in R \\
&x_i \geq 0, \forall i \in C_0 \\
&y_i \geq 0, \forall i \in C_1 \\
&z_i \geq 0, \forall i \in C_2 \\
&t \geq 0
\end{aligned}
$$

Again, for the social function SUM_{perc} it suffices to delete the two terms $-2\gamma O'_rO''_r$ and $2E'_rE''_r$ occurring at the end of the dual constraints. We give the following upper bounds.

Theorem 3. *For each bidimensional linear congestion game* $(\mathcal{G},\mathcal{C})$*, it holds* $PoS(\mathcal{G},\mathcal{C}) \leq 1+\frac{2}{\sqrt{7}}$ *under the social function* SUM_{pres} *and* $PoS(\mathcal{G},\mathcal{C}) \leq 2.92$ *under the social function* SUM_{perc}*.*

Proof. For the social function SUM_{pres}, set $\gamma = 1+\frac{2}{\sqrt{7}}$, $x_i = \frac{1}{\sqrt{7}}$ for any $i \in C_0$, $y_i = \frac{1}{\sqrt{7}}$ for any $i \in C_1$, $z_i = \frac{1}{\sqrt{7}}$ for any $i \in C_2$ and $t = \frac{1}{2}+\frac{1}{2\sqrt{7}}$. With these values, for any $r \in R$, the dual constraint becomes

$$E_r^2(3-\sqrt{7}) - E_r(2O_r - 1 - \sqrt{7}) + 2E'_rE''_r(\sqrt{7}-3) + 2(E'_rO''_r + E''_rO'_r)$$

$$+(O_r^2 - O_r)(3+\sqrt{7}) - 2O'_rO''_r(3+\sqrt{7}) \geq 0.$$

The claim follows by applying Lemma 3.
For the social function SUM_{perc}, set $\gamma = 2.92$, $x_i = 0.68$ for any $i \in C_0$, $y_i = 1.3$ for any $i \in C_1$, $z_i = 1.3$ for any $i \in C_2$ and $t = 0.81$. With these values, for any $r \in R$, the dual constraint becomes

$$49E_r^2 + E_r(62E'_r + 62E''_r - 68O_r - 62O'_r - 62O''_r + 81) + 130E'_rO''_r + 130E''_rO'_r$$

$$-422E'_rE''_r + 211O_r^2 - 149O_r + 2(81O'_rO''_r - 31O'_r - 31O''_r) \geq 0.$$

The claim follows by applying Lemma 4. □

For these cases, unfortunately, we are not able to devise matching lower bounds. The following result is obtained by suitably extending the lower bounding instance given in [10] for the price of stability of congestion games.

Theorem 4. *For any* $\epsilon > 0$*, there exist two bidimensional linear congestion games* $(\mathcal{G},\mathcal{C})$ *and* $(\mathcal{G}',\mathcal{C}')$ *such that* $PoS(\mathcal{G},\mathcal{C}) \geq \frac{1+\sqrt{5}}{2} - \epsilon$ *under the social function* SUM_{pres} *and* $PoS(\mathcal{G}',\mathcal{C}') \geq \frac{5+\sqrt{17}}{4} - \epsilon$ *under the social function* SUM_{perc}*.*

References

1. Aland, S., Dumrauf, D., Gairing, M., Monien, B., Schoppmann, F.: Exact Price of Anarchy for Polynomial Congestion Games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 218–229. Springer, Heidelberg (2006)
2. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. SIAM Journal of Computing 38(4), 1602–1623 (2008)
3. Awerbuch, B., Azar, Y., Epstein, A.: The Price of Routing Unsplittable Flow. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 57–66. ACM Press (2005)
4. Bhawalkar, K., Gairing, M., Roughgarden, T.: Weighted Congestion Games: Price of Anarchy, Universal Worst-Case Examples, and Tightness. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part II. LNCS, vol. 6347, pp. 17–28. Springer, Heidelberg (2010)
5. Bilò, V.: A Unifying Tool for Bounding the Quality of Non-Cooperative Solutions in Weighted Congestion Games. CoRR abs/1110.5439 (2011)

6. Bilò, V., Fanelli, A., Flammini, M., Moscardelli, L.: Graphical Congestion Games. Algorithmica 61(2), 274–297 (2011)
7. Caragiannis, I., Flammini, M., Kaklamanis, C., Kanellopoulos, P., Moscardelli, L.: Tight Bounds for Selfish and Greedy Load Balancing. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part I. LNCS, vol. 4051, pp. 311–322. Springer, Heidelberg (2006)
8. Christodoulou, G., Koutsoupias, E.: The Price of Anarchy of Finite Congestion Games. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 67–73. ACM Press (2005)
9. Christodoulou, G., Koutsoupias, E.: On the Price of Anarchy and Stability of Correlated Equilibria of Linear Congestion Games. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 59–70. Springer, Heidelberg (2005)
10. Christodoulou, G., Koutsoupias, E., Spirakis, P.G.: On the Performance of Approximate Equilibria in Congestion Games. Algorithmica 61(1), 116–140 (2011)
11. Fotakis, D., Gkatzelis, V., Kaporis, A.C., Spirakis, P.G.: The Impact of Social Ignorance on Weighted Congestion Games. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 316–327. Springer, Heidelberg (2009)
12. Fotakis, D., Kontogiannis, S., Spirakis, P.: Selfish Unsplittable Flows. Theoretical Computer Science 348, 226–239 (2005)
13. Koutsoupias, E., Papadimitriou, C.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
14. Harks, T., Klimm, M.: On the Existence of Pure Nash Equilibria in Weighted Congestion Games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 79–89. Springer, Heidelberg (2010)
15. Harks, T., Klimm, M., Möhring, R.H.: Characterizing the Existence of Potential Functions in Weighted Congestion Games. In: Mavronicolas, M., Papadopoulou, V.G. (eds.) SAGT 2009. LNCS, vol. 5814, pp. 97–108. Springer, Heidelberg (2009)
16. Rosenthal, R.W.: A Class of Games Possessing Pure-Strategy Nash Equilibria. International Journal of Game Theory 2, 65–67 (1973)

Mobile Network Creation Games*

Michele Flammini[1], Vasco Gallotti[1], Giovanna Melideo[1], Gianpiero Monaco[1], and Luca Moscardelli[2]

[1] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
{michele.flammini,vasco.gallotti, giovanna.melideo,gianpiero.monaco}@univaq.it
[2] Department of Economic Studies, University of Chieti-Pescara, Italy
luca.moscardelli@unich.it

Abstract. We introduce a new class of network creation games, called *mobile network creation games*, modelling the spontaneous creation of communication networks by the distributed and uncoordinated interaction of k selfish mobile devices. Each device is owned by a player able to select a node in an underlying positions graph so as to minimize a cost function taking into account two components: the distance from her home position, and the number of players she is not connected to, with the connectivity costs being prevailing, i.e., the Nash Equilibria are stable solution states in which communication is possible among all the players. We show that the game always admits a Pure Nash equilibrium, even if the convergence after a finite number of improving movements is guaranteed only when players perform their best possible moves. More precisely, if initial positions are arbitrary, that is not necessarily coinciding with the home ones, an order of kD best moves is necessary (and sufficient) to reach an equilibrium, where D is the diameter of the positions graph. As for the Nash equilibria performances, we first prove that the price of stability is 1 (i.e. an optimal solution is also a Nash equilibrium). Moreover, we show that the lack of centralized control of mobile devices is a major issue in terms of final performance guaranteed. Namely, the price of anarchy is $\Theta(kD)$. Nevertheless, we are able to prove that if players start at their home positions, in $\Theta(k \min\{k^2, D\})$ best moves they reach an equilibrium approximating the optimal solution by a factor of $\Theta(k \min\{k, D\})$.

Keywords: Network Creation Games, Price of Anarchy, Price of Stability, Speed of Convergence.

1 Introduction

The emerging global communication and service infrastructures like the Internet are characterized by decentralization, autonomy, and general lack of

* This research was partially supported by the PRIN 2008 research project COGENT (COmputational and GamE-theoretic aspects of uncoordinated NeTworks), funded by the Italian Ministry of University and Research.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 159–170, 2012.

coordination among the heterogeneous network entities, the network in turn intrinsically being a common playground for a large number of users. These users exhibit various degrees of intentional or unintentional non cooperative behavior, while competing for shared and often scarce resources. Besides, they increasingly demand stable connection and seamless access to the network resources, by means of the enormous potential offered by the new wireless equipments and capabilities. The combination of uncoordination and wireless access induces a degree of dynamism never experienced before, that must be necessarily faced by the emerging services and applications, and calls for a pressing solution of the resulting scientific and technological challenges.

In such a highly elusive and mutable setting, the general mismatch between the network optimization goals and the competing users private interests motivated an extensive research on (algorithmic) game theoretical frameworks aiming to characterize the system outcome by suitable stable solution concepts like the Nash Equilibrium [17]. A strategy profile for the players is a Nash Equilibrium if no player can gain by unilaterally deviating from her own strategy choice. One of the main tools for evaluating the degradation of the system performance induced by the lack of coordination of selfish players is the price of anarchy (PoA) [15,18], a measure that compares the social cost of the worst case Nash equilibrium to the social optimum one. A related optimistic measure for evaluating the cost of the best possible equilibrium is the price of stability (PoS) [3], a measure that compares the social cost of the best Nash equilibrium to the social optimum one. The complexity of computing a Nash equilibrium and the speed of convergence to such solutions have been extensively studied for many classes of non-cooperative games, like in [1,10] in the context of congestion games.

The first game-theoretical model for the spontaneous construction of networks by means of the distributed and uncoordinated interaction of many autonomous players has been proposed in [14]. Moreover, in [11], the authors investigated network creation games where selfish nodes (the players) pay for the links that they establish towards the other nodes and benefit from decreasing the shortest paths lengths to all destinations. In such a setting, they characterized the existence of Nash equilibria and derived corresponding bounds on the PoA. Many subsequent papers conducted similar studies in different related network creation settings under various assumptions [2,6,7,16], even if to the best of our knowledge they did not consider mobility aspects.

An interesting class of mobile network creation problems recently proposed in the literature is the one of the so-called *movement problems*, in which the goal is that of finding the positions of k devices which achieve a global property of the induced subnetwork, minimizing the maximum or total movement [5]. A particularly relevant case is the one in which the aim is that of obtaining complete connectivity, i.e. all-to-all multi-hop communication among the mobile devices, while minimizing the total movement. Assuming $P \neq NP$, in [5] almost tight bounds for such a problem have been provided: an $\Omega(|V|^{1-\varepsilon})$-inapproximability result and an $O(\min\{|V|\log|V|, k\})$-approximation algorithm. Movement

problems have then attracted further research due to the interesting modeled real world scenarios [8,13], but only under a classical centralized setting.

Some fairly related game-theoretical studies concern Voronoi [9,12] and Isolation games [4,19]. In fact, they focus on the positioning on graphs of interfering players interested in being faraway according to different metrics. However, such games do not deal with network creation.

We investigate the movement problems of [5] in a non-cooperative scenario in which the mobile users/devices are autonomous, thus creating a bridge with the extensive area of the network creation games. More precisely, we consider a collection $\{1, \ldots, k\}$ of selfish mobile devices (or players) able to move on the nodes of a given positions graph $G = (V, E)$. Each device i has an associated home location $h_i \in V$; once positions are selected she is "directly" connected to all the devices at distance smaller than a given threshold r, while she can communicate to the remaining ones in a multi-hop fashion. The cost of each player has two components: (i) the distance from her home location, plus (ii) α times the number of players she cannot reach via multi-hop communication, where $\alpha > 0$ is a given parameter quantifying the connectivity contribution on the cost incurred by each player. Since we are interested in solutions (i.e., equilibria) connecting all the players, as we will show in the next section, we have to focus on specific values of α, namely $\alpha > D - r$, with D being the diameter of G. Each device is interested in selecting a position (strategy) minimizing her own cost. The social cost of a solution is simply the sum of all players' costs. Notice that the final social cost coincides with the sum of the distances of the players from their home locations.

It is worth emphasizing that our model can be applied to a wide variety of contexts; for instance, the devices could be equipped with radio antennas able to receive and transmit data within a range of r edges of the network G, or with wired (for instance optical) transmitters and receivers able to transmit data to the other devices being far at most r edges in G.

We show that the game always admits a Pure Nash equilibrium, even if the convergence after a finite number of improving movements is guaranteed only when players perform their best possible moves. More precisely, if initial positions are arbitrary, that is not necessarily coinciding with the home ones, an order of kD best moves is necessary (and sufficient) to reach an equilibrium. As for the Nash equilibria performances, we first prove that the price of stability is 1 (i.e. an optimal solution is also a Nash equilibrium). Moreover, we show that uncentralized control of mobile devices is a major issue in terms of final performance guaranteed, as the price of anarchy is $\Theta(kD)$. Nevertheless, we are able to prove that if players start at their home positions, in $\Theta(k \min\{k^2, D\})$ best moves they reach an equilibrium approximating the optimal solution by a factor of $\Theta(k \min\{k, D\})$. In the sequel we refer to Pure Nash Equilibria as Nash Equilibria.

The remainder of this paper is organized as follows. In the next section we provide the basic notation and definitions. Section 3 concerns the existence and convergence to Nash equilibria; Section 4 presents the results on the price of

anarchy. Finally, Section 5 gives some conclusive remarks and outlines some interesting open questions.

Due to space limitations, some proofs are omitted. They will appear in the full version of this paper.

2 Model and Preliminaries

Given an undirected connected graph $G = (V, E)$, for any pair of nodes $u, v \in V$, let $d(u, v)$ be the length of a shortest path connecting u and v in G. Let $D = \max_{u,v \in V} d(u, v)$ be the diameter of graph G.

A *mobile network creation game* is defined by an undirected graph $G = (V, E)$ and a set of k players $[k] = \{1, \ldots, k\}$ aiming at selfishly minimizing their own cost. The *strategy set* of each player is given by the set V of all the nodes of G and the set of strategy profiles is $\mathcal{S} := V^k$, where a *state* or *strategy profile* $S \in \mathcal{S}$ is a k-tuple $S = (s_1, \ldots, s_k)$ such that, for any $i \in [k]$, $s_i \in V$ is the node chosen by player i in S. Each player $i \in [k]$ has an *home position* $h_i \in V$. The *home state* $H = (h_1, \ldots, h_k)$ is the state in which every player is selecting her own home position.

For each $i \in [k]$ and $j \in [k] \setminus \{i\}$, $r_{i,j}$ is the threshold defining the maximum distance between the positions of i and j needed to allow communication from i to j. Given a strategy profile $S = (s_1, \ldots, s_k)$, we say that player j is directly connected to player i (i.e. player i can send data to player j), if $d(s_i, s_j) \leq r_{i,j}$. Multi-hop connections are allowed and therefore player j is connected to player i either if such a connection is direct, or if there exist a player l such that l is connected to player i, and player j is connected to player l.

Given a strategy profile $S = (s_1, \ldots, s_k)$, for any $i \in [k]$, let $K_i(S) \subseteq [k]$ be the set of the players connected to player i (including herself), and $k_i(S) = |K_i(S)| \geq 1$; moreover, let $k_{max}(S) = \max_{i \in [k]} k_i(S)$ be the biggest cardinality of a subset of connected players in state S. The cost function of player i is:

$$c_i(S) = \alpha(k - k_i(S)) + d(s_i, h_i) \tag{1}$$

where $\alpha > 0$ is a scaling factor determining the connectivity influence on the cost incurred by player i, and $C(S) = \sum_{i \in [k]} c_i(S)$ is the social cost associated to strategy profile S.

For any k-tuple A, let (A_{-i}, x) denote the k-tuple obtained from A by replacing its i-th element with x. Given a strategy profile $S = (s_1, \ldots, s_k)$, an *improving move* of player i in S is a strategy $u \in V$ such that $c_i((S_{-i}, u)) < c_i(S)$. A strategy profile is a *Nash equilibrium* if and only if no player can perform any improving move. A game is said to be *convergent* if, given any initial state S, any sequence of improving moves leads to a Nash equilibrium.

Furthermore, a *best move* of player i in S is the best improvement move available to i in S, i.e. an improvement move $u \in V$ for i in S such that $c_i((S_{-i}, u)) \leq c_i((S_{-i}, v))$ for any other improvement move $v \in V$ for i in S. A *best move dynamics* is a sequence of best moves.

We are interested in bounding the performances of Nash equilibria with respect to the social optimum OPT, i.e. the social value of a strategy profile S^* such that $C(S^*) = \min_{S \in \mathcal{S}} C(S)$.

Let $\mathcal{N}$ denote the set of the Nash equilibria; the *price of stability* is the best case ratio between the social optimum and the social value of a Nash equilibrium, i.e., $\text{PoS} = \inf_{S \in \mathcal{N}} \frac{C(S)}{\text{OPT}}$; the *price of anarchy* is the worst case ratio between the social value of a Nash equilibrium and the social optimum, i.e., $\text{PoA} = \sup_{S \in \mathcal{N}} \frac{C(S)}{\text{OPT}}$. Moreover, if $\mathcal{N}_{\mathcal{H}}$ is the set of the Nash equilibria achievable only by best move dynamics starting from the home state, the *home price of anarchy* is the worst case ratio between the social value of such a Nash equilibrium and the social optimum, i.e. $\text{PoA}_H = \sup_{S \in \mathcal{N}_{\mathcal{H}}} \frac{C(S)}{\text{OPT}}$. Clearly, since $\mathcal{N}_{\mathcal{H}} \subseteq \mathcal{N}$, $\text{PoA}_H \leq \text{PoA}$.

Since we are interested in connecting all the players, we have to restrict our attention to the values of α being greater than $D - r_{min}$, with $r_{min} = \min_{i,j \in [k]} r_{i,j}$. In fact, the following claim holds.

Claim. All Nash Equilibria of a given Mobile Network Creation Game are such that all the players are connected if and only if $\alpha > D - r_{min}$.

In the following, even if not explicitly stated, we assume $r_{i,j} = 1$ for all $i \in [k]$ and $j \in [k] \setminus \{i\}$; nevertheless, all the results extend to the more general case in which, for all $i \in [k]$ and $j \in [k] \setminus \{i\}$, $r_{i,j} = r \geq 1$, as mentioned in Section 5. Notice that when $r_{i,j} = r \geq 1$ for all $i \in [k]$ and $j \in [k] \setminus \{i\}$, for any state S if $j \in K_i(S)$, then also $i \in K_j(S)$. That is, i is connected to j if and only if j is connected to i. Finally, we restrict our attention to the non trivial instances in which for the home state H it does not hold that $k_i(H) = k$ for all $i \in [k]$. Therefore, the inequality $C(S) \geq \text{OPT} \geq 1$ always holds.

3 Nash Equilibria: Existence and Convergence

In this section we first show that any optimal solution to the Mobile Network Creation Games is also a Nash equilibrium. It clearly implies that the price of stability is 1. Moreover we prove that convergence is guaranteed only restricting to best move dynamics. In particular, we provide asymptotically tight bounds on the best moves needed to reach an equilibrium.

Theorem 1. *Any optimal solution to the Mobile Network Creation Games is also a Nash equilibrium. Moreover Mobile Networks Creation Games are not convergent.*

Let us now focus on best move dynamics. The selfish behavior of the mobile players can be modelled by a directed *Nash (Best Move) Dynamics Graph* $\mathcal{B} = (\mathcal{S}, A)$, where each vertex corresponds to a strategy profile and there is an arc $(S, S') \in A$ with label i if and only if $S' = (S_{-i}, u)$ and there exists a best move for player i in state S in which i selects strategy $u \in V$.

Lemma 1. *$k_i(S') = k_{max}(S') \geq k_{max}(S)$ for any arc $(S, S') \in A$ with label i. Moreover, $K_i(S') = K_j(S')$ for any other $j \in [k]$ such that $k_j(S') = k_i(S') = k_{max}(S')$, i.e. after any best move leading from state S to state S' the subset of connected players of cardinality $k_{max}(S')$ is unique.*

Lemma 2. *For any arc $(S, S') \in A$ with label i, if $k_{max}(S') = k_{max}(S)$ then player i is not increasing the number of players she is connected to, i.e. $k_i(S') = k_i(S)$.*

As we will see in the following, speed of convergence and price of anarchy differ if players start at the home state H or from any arbitrary state S.

Theorem 2. *Mobile Network Creation Games starting from a generic state S converge to a Nash equilibrium in $O(kD)$ best moves.*

The following theorem shows that the upper bound of the above theorem is tight.

Theorem 3. *There exists a Mobile Network Creation Game starting from a state S and converging to a Nash equilibrium in $\Omega(kD)$ best moves.*

Proof. Let us consider the graph depicted in Figure 1 in which the nodes labeled h_i for $i = 1, \ldots, k$ are the home position of players $1, \ldots, k$, respectively. First of all, notice that the graph nodes can be partitioned into $l + 1$ rows, and that the diameter D of the graph, for every value of k and l, is always between l and $2l$. A *round* is a sequence of best moves in which each player moves once in increasing order, i.e. from player 1 to player k. If the dynamics starts from the initial state $S = (s_1, \ldots, s_k)$ and players move in increasing order (from player 1 to player k), it is easy to see that after a round, all the players move from the first (i.e. the topmost) row to the second row; more generally, for $j = 1, \ldots, l - 2$, after the j-th round, all the players move to the $j + 1$-th row, in order to decrease the distance from their own home position. Thus, after $l - 1$ rounds, the player reach a Nash equilibrium in which all players are in the row above the home vertices. Therefore, $(l - 1) \cdot k$ best moves are needed in order to reach an equilibrium and a bound of $\Omega(kD)$ best moves holds for the convergence. □

Better bounds hold for dynamics starting from the home state. On this respect, let us first prove the following useful lemma.

Lemma 3. *Consider a best move dynamics starting from the home state H. For any arc $(S, S') \in A$ corresponding to a best move of player i, there exists a player $j \in K_i(S')$ such that $s'_j = h_j$, i.e. player j is selecting in state S' her home position.*

Proof. Let $H = S^0, S^1, S^2, \ldots, S^T$ the states corresponding to the considered dynamics and $p_t \in [k]$, $t = 1, \ldots, T$, the player moving from state S^{t-1} to state S^t; in other words, S^t is the state reached after the t-th best move of the dynamics by a best move of player p_t.

By Lemma 1, we know that, for any $t = 1, \ldots, T$, $K_{p_t}(S^t)$ is the unique biggest subset of $k_{max}(S^t)$ players connected in S^t. For any $t = 1, \ldots, T$, let a_t be the

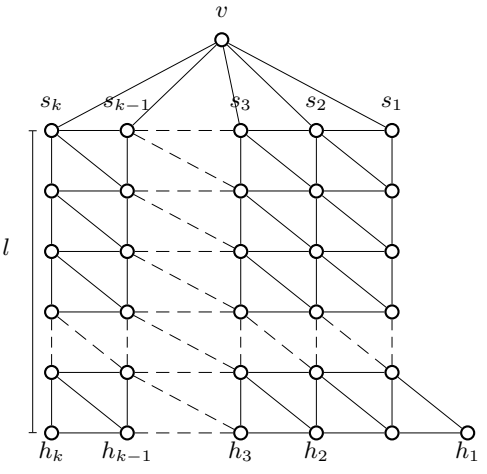


Fig. 1. A Mobile Network Creation Game in which $\Omega(kD)$ best moves are needed in order to reach a Nash Equilibrium

number of players in $K_{p_t}(S^t)$ whose strategy in S^t is their own home position, and let $\bar{a}_t$ be $|K_{p_t}(S^t)| - a_t$. Furthermore, let $\bar{b}_t$ be the number of players not belonging to $K_{p_t}(S^t)$ and whose strategy in S^t is not their own home position. Notice that, for any $t = 1, \ldots, T$, the number of players not being on their own home position in state S^t is $\bar{a}_t + \bar{b}_t$.

We first show that $a_t - a_{t-1} \geq \bar{b}_t - \bar{b}_{t-1}$ for any $t = 2, \ldots, T$. We distinguish between two disjoint cases:

- If $s_t^{t-1} \neq h_t$, i.e. player p_t is not selecting in state S^{t-1} her own home position, the number of players not selecting in state S^t their own home position cannot increase with respect to the one of state S^{t-1}, i.e.

$$\bar{a}_t + \bar{b}_t \leq \bar{a}_{t-1} + \bar{b}_{t-1}. \tag{2}$$

  Moreover, by Lemma 1,

$$a_t + \bar{a}_t \geq a_{t-1} + \bar{a}_{t-1}. \tag{3}$$

  By combining equations 2 and 3, it follows that $a_t - a_{t-1} \geq \bar{b}_t - \bar{b}_{t-1}$.
- If $s_t^{t-1} = h_t$, i.e. player p_t is selecting in state S^{t-1} her own home position, the number of players not selecting in state S^t their own home position can increase of at most 1 with respect to the one of state S^{t-1}, i.e.

$$\bar{a}_t + \bar{b}_t \leq \bar{a}_{t-1} + \bar{b}_{t-1} + 1. \tag{4}$$

  We now show that

$$a_t + \bar{a}_t \geq a_{t-1} + \bar{a}_{t-1} + 1. \tag{5}$$

  If p_t is such that $K_{p_t}(S^{t-1}) = K_{p_{t-1}}(S^{t-1})$, i.e. player p_t belongs in state S^{t-1} to the unique biggest subset of connected players, it must hold that

$|K_{p_t}| > |K_{p_{t-1}}|$ because player p_t move from her own home position only if she increases the number of players she is connected to. Otherwise, i.e. player p_t does not belong in state S^{t-1} to the unique biggest subset of connected players, then also in this case we can claim that $|K_{p_t}| > |K_{p_{t-1}}|$ because one of the strategies available for player p_t is that of choosing a node making her connected to the $|K_{p_{t-1}}|$ players constituting the biggest subset of connected players in state S^{t-1}.
By combining equations 4 and 5, it follows that $a_t - a_{t-1} \geq \bar{b}_t - \bar{b}_{t-1}$.

Notice that, since after the first best move of the dynamics only player p_1 is not on her own home position, it follows that $a_1 \geq 1$ and $\bar{b}_1 = 0$.

We now show that, for every $t = 2, \ldots, T$, $a_t \geq 1$: Consider a generic $t = 2, \ldots, T$; for any $t' = 2, \ldots, t$ it holds that $a_{t'} - a_{t'-1} \geq \bar{b}_{t'} - \bar{b}_{t'-1}$. By summing over all such $t-1$ inequalities, it follows that $a_t - a_1 \geq \bar{b}_t - \bar{b}_1$. Therefore, $a_t \geq a_1 + \bar{b}_t - \bar{b}_1 \geq 1$ because $a_1 \geq 1$, $\bar{b}_1 = 0$ and $\bar{b}_t \geq 0$. □

Theorem 4. *Mobile Network Creation Games starting from the home state H converge to a Nash equilibrium in $O(k \cdot \min\{k^2, D\})$ best moves.*

Proof. By Theorem 2, we already know that every game converges in at most $O(kD)$ best moves. In order to prove the $O(k \cdot \min\{k^2, D\})$ bound, we now show that every game starting from the home state H converges to a Nash equilibrium in $O(k^3)$ best moves.

Consider the potential function $\Phi : \mathcal{S} \to \mathbb{N} \setminus \{0\}$ defined as follows:

$$\Phi(S) = (k - k_{max}(S))(2k^2 - k + 1) + \sum_{i \in [k]} \sum_{j \in [k]} \Delta_{i,j}(S),$$

where $\Delta_{i,j}(S) = \min\{2k-1, \max\{0, d(s_i, h_i) - d(x_{i,j}, h_i)\}\}$, $x_{i,j}$ being the node of the graph minimizing its distance from h_i and such that $d(x_{i,j}, h_j) \leq k-1$.

Consider the (Best Move) Nash Dynamics Graph $\mathcal{B} = (\mathcal{S}, A)$ associated to a given Mobile Network Creation Game. If $\Phi(S) > \Phi(S') > 0$ for any $(S, S') \in A$, it follows that such a game converges to a Nash equilibrium in $O(k^3)$ best moves, because the maximum value the function can assume is $(k-1)(2k^2 - k + 1) + k^2(2k-1) = O(k^3)$. In order to complete the upper bound analysis, we have to prove that $\Phi(S) > \Phi(S')$ for any $(S, S') \in A$.

By Lemma 1, for any arc $(S, S') \in A$, $k_{max}(S') \geq k_{max}(S)$; we divide the proof into two disjoint cases:

- if $k_{max}(S') = k_{max}(S)$, by Lemma 2 it must hold that $k_i(S') = k_i(S)$. In such a case, it is easy to verify that

$$\Phi(S') - \Phi(S) = \sum_{j \in [k]} \left(\Delta_{i,j}(S') - \Delta_{i,j}(S)\right).$$

  Since player i is decreasing her own distance from the home position, i.e. $d(s'_i, h_i) < d(s_i, h_i)$, for every $j \in [k]$, $\Delta_{i,j}(S') - \Delta_{i,j}(S) \leq 0$. Moreover, we know by Lemma 3 that there exists at least a player $p \in [k]$ being in S'

on her own home position and such that $K_p(S') = K_i(S')$; if we consider the term of the summation for which $j = p$, it is possible to show that $\Delta_{i,p}(S') - \Delta_{i,p}(S) \leq -1$. First of all, we notice that $\Delta_{i,p}(S) > 0$, because otherwise already in state S player i was close to h_i at least as node $x_{i,p}$ and thus, by recalling the definition of $x_{i,p}$, she cannot be connected to player p in state S'. Moreover, the following property holds: Since $i \in K_p(S')$, i.e. player i is connected to player p in state S', $d(s'_i, h_p) \leq k-1$, otherwise the multi-hop connection between i and p cannot be established. It follows that $d(s'_i, h_i) \leq d(s'_i, h_p) + d(h_p, x_{i,p}) + d(x_{i,p}, h_i) \leq 2(k-1) + d(x_{i,p}, h_i)$, where the first inequality holds by applying twice the triangular inequality and the second inequality by the above stated property and the definition of $x_{i,p}$; therefore, $\Delta_{i,p}(S') \leq d(s'_i, h_i) - d(x_{i,p}, h_i) \leq 2(k-1)$ and if $\Delta_{i,p}(S) = 2k-1$ we obtain $\Delta_{i,p}(S') - \Delta_{i,p}(S) \leq 2(k-1) - (2k-1) = -1$. It remains to analyze the case in which $0 < \Delta_{i,p}(S) < 2k-1$; in such a case, since $d(s'_i, h_i) < d(s_i, h_i)$, it trivially follows that $\Delta_{i,p}(S') - \Delta_{i,p}(S) \leq -1$.
As a consequence, $\Phi(S') - \Phi(S) = \sum_{j \in [k]} (\Delta_{i,j}(S') - \Delta_{i,j}(S)) \leq -1$.

– if $k_{max}(S') > k_{max}(S)$, then

$$\Phi(S') - \Phi(S) \leq -(2k^2 - k + 1) + \sum_{j \in [k]} (\Delta_{i,j}(S') - \Delta_{i,j}(S)) .$$

Clearly, $\sum_{j \in [k]} (\Delta_{i,j}(S') - \Delta_{i,j}(S))$ is at most $\sum_{j \in [k]} \Delta_{i,j}(S') \leq k(2k-1)$. Therefore, $\Phi(S') - \Phi(S) \leq -(2k^2 - k + 1) + k(2k-1) = -1$.

□

The following theorem shows that the upper bound of the above theorem is tight.

Theorem 5. *There exists a Mobile Network Creation Game starting from the home state H that converges to a Nash equilibrium in $\Omega(k \cdot \min\{k^2, D\})$ best moves.*

4 Price of Anarchy

In this section we provide matching upper and lower bounds on the price of anarchy and on the home price of anarchy of mobile network creation games.

Theorem 6. *The price of anarchy of Mobile Network Creation Games is $\Theta(kD)$.*

Fortunately, for the home price of anarchy we are able to show a better tight bound, only depending on the number of players.

Theorem 7. *The home price of anarchy of Mobile Network Creation Games is $\Theta(k \min\{k, D\})$.*

Proof. By Theorem 6, we already know that the price of anarchy is $O(kD)$; since it trivially holds that $\text{PoA}_H \leq \text{PoA}$, it also holds that $\text{PoA}_H = O(kD)$. In order to prove the $O(k \cdot \min\{k, D\})$ bound, we now show that the home price of anarchy is $O(k^2)$.

Let $d_{max} = \max_{i,j \in [k]}(h_i, h_j)$ be the maximum distance between the home position of two players. It is possible to show that $\text{OPT} \geq \max\{1; d_{max} - k\}$; by definition, $\text{OPT} \geq 1$. Let i and j the players such that $d(h_i, h_j) = d_{max}$. By the triangular inequality, $d(h_i, s_i^*) + d(s_i^*, s_j^*) + d(s_j^*, h_j) \geq d(h_i, h_j) = d_{max}$. Hence, since $d(s_i^*, s_j^*) \leq k - 1$ because i and j are connected in S^*, we obtain that $d(h_i, s_i^*) + d(s_j^*, h_j) \geq d_{max} - k + 1$.

Furthermore, given a Nash Equilibrium S reached from the home position by performing best moves, since by Lemma 3 there exists at least a player j such that $s_j = h_j$, i.e. a player selecting her own home position at equilibrium, every player has to be connected to player j and, for every $i = 1, \ldots, k$, $d(s_i, h_j) \leq k - 1$. Since, for every $i = 1, \ldots, k$, $d(h_i, h_j) \leq d_{max}$, it follows by the triangular inequality that $d(s_i, h_i) \leq d(s_i, h_j) + d(h_j, h_i) \leq k - 1 + d_{max}$. Therefore, $C(S) = \sum_{i \in [k] \setminus \{j\}} d(s_i, h_i) \leq \sum_{i \in [k] \setminus \{j\}} (k - 1 + d_{max}) = (k-1)(k-1+d_{max})$.

We now distinguish between 2 disjoint cases:

- If $d_{max} \leq 3k$, we obtain that $\text{OPT} \geq 1$ and $C(S) \leq (k-1)(4k-1)$. Therefore, it follows that $\text{PoA} = O(k^2)$.
- If $d_{max} > 3k$, by simple algebraic calculation, we obtain that $\text{PoA} \leq \frac{(k-1)(k-1+d_{max})}{d_{max}-k} \leq 2k^2 = O(k^2)$.

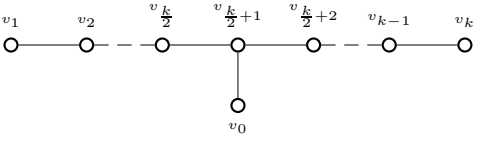


Fig. 2. A Mobile Network Creation Game having home price of anarchy $\Omega(k^2)$

In order to show the $\Omega(k^2)$ lower bound, consider the instance depicted in Figure 2, in which k is such that $k \mod 4 = 0$.

Let $f : \mathbb{N} \to \{1, \ldots, k\}$ be a function defined as follows:

$$f(n) = \begin{cases} \frac{k}{2} & \text{if } n = 0 \\ \frac{k}{2} + n + 1 & \text{if } n \text{ is odd} \\ \frac{k}{2} - n & \text{if } n \text{ is even, } n > 0 \end{cases}$$

Let $h_i = v_i$ for any $i \in [k] \setminus \left\{\frac{k}{2}\right\}$, and $h_{\frac{k}{2}} = v_0$. Consider the evolution starting from the state $H = (h_1, \ldots, h_k)$ and in which, for $i = 1, \ldots, \frac{k}{2} - 1$, the i-th best move is performed by player $f(i)$, moving from her home position $v_{f(i)}$ to position $v_{f(i-1)}$. Notice that $k_{max}(H) = \frac{k}{2} + 1$, and that after each best move k_{max} increases by 1, till reaching a Nash equilibrium S after the $\frac{k}{2} - 1$-th

best move, such that $k_{max}(S) = k$. Moreover, it is easy to check that in such an evolution each player performs at most 1 best move; hence, for $i = 1, \ldots, \frac{k}{2} - 1$, $d(s_i, h_i) = |f(i) - f(i-1)| = 2i$. Therefore, $C(S) \geq \sum_{i=1}^{\frac{k}{2}-1} 2i = \Omega(k^2)$. □

An example for $k = 8$ is shown in Figure 3.

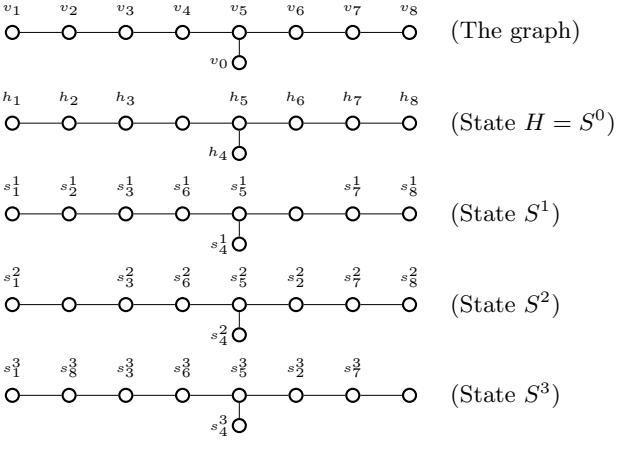


Fig. 3. An example showing the lower bound construction for $k = 8$. The equilibrium is reached in state S^3.

5 Extensions and Conclusions

We have considered mobile network creation games according to the model of [5]. All the results are given for transmission ranges $r_{i,j} = 1$ for any $i \in [k]$ and $j \in [k] \setminus \{i\}$.

However, even if we do not claim it explicitly, they extend to any $r_{i,j} = r \geq 1$ as follows. The $\Theta(kD)$ bounds on rate of convergence of best move dynamics starting from generic states and on the price of anarchy still hold. On the other hand, if players start from their home locations, the rate of convergence becomes $\Theta(k \cdot \min\{k^2 r, D\})$, and the home price of anarchy $\Theta(k \cdot \min\{kr, D\})$. All the details will be given in the full version of the paper.

Many interesting problems are left open. First of all, what about non-uniform transmission thresholds? A worth investigating issue would be that of considering different communication patterns, such as multicasting or many to many.

References

1. Ackermann, H., Röglin, H., Vöcking, B.: On the impact of combinatorial structure on congestion games. J. ACM 55(6) (2008)
2. Alon, N., Demaine, E.D., Hajiaghayi, M.T., Leighton, T.: Basic network creation games. In: SPAA, pp. 106–113. ACM (2010)

3. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: FOCS, pp. 295–304. IEEE Computer Society (2004)
4. Biló, V., Flammini, M., Monaco, G., Moscardelli, L.: On the performances of nash equilibria in isolation games. J. Comb. Optim. 22(3), 378–391 (2011)
5. Demaine, E.D., Hajiaghayi, M.T., Mahini, H., Sayedi-Roshkhar, A.S., Oveisgharan, S., Zadimoghaddam, M.: Minimizing movement. ACM Transactions on Algorithms 5(3) (2009)
6. Demaine, E.D., Hajiaghayi, M.T., Mahini, H., Zadimoghaddam, M.: The price of anarchy in network creation games. In: PODC, pp. 292–298. ACM (2007)
7. Demaine, E.D., Hajiaghayi, M.T., Mahini, H., Zadimoghaddam, M.: The price of anarchy in cooperative network creation games. In: STACS, pp. 301–312 (2009)
8. Demaine, E.D., Hajiaghayi, M.T., Marx, D.: Minimizing Movement: Fixed-Parameter Tractability. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 718–729. Springer, Heidelberg (2009)
9. Dürr, C., Thang, N.K.: Nash Equilibria in Voronoi Games on Graphs. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 17–28. Springer, Heidelberg (2007)
10. Fabrikant, A., Papadimitriou, C.H., Talwar, K.: The complexity of pure nash equilibria. In: STOC, pp. 604–612. ACM (2004)
11. Fabrikant, A., Luthra, A., Maneva, E., Papadimitriou, C.H., Shenker, S.: On a network creation game. In: PODC, pp. 347–351. ACM (2003)
12. Feldmann, R., Mavronicolas, M., Monien, B.: Nash Equilibria for Voronoi Games on Transitive Graphs. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 280–291. Springer, Heidelberg (2009)
13. Friggstad, Z., Salavatipour, M.R.: Minimizing movement in mobile facility location problems. In: FOCS, pp. 357–366. IEEE Computer Society (2008)
14. Jackson, M.O., Wolinsky, A.: A strategic model of social and economic networks. Journal of Economic Theory 71(1), 44–74 (1996)
15. Koutsoupias, E., Papadimitriou, C.H.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
16. Leonardi, S., Sankowski, P.: Network formation games with local coalitions. In: PODC, pp. 299–305. ACM (2007)
17. Nash, J.: Non-cooperative games. The Annals of Mathematics 54(2), 286–295 (1951)
18. Papadimitriou, C.H.: Algorithms, Games, and the Internet (Extended Abstract). In: Yu, Y., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 1–3. Springer, Heidelberg (2001)
19. Zhao, Y., Chen, W., Teng, S.-H.: The isolation game: A game of distances. Theor. Comput. Sci. 410(47-49), 4905–4919 (2009)

Homonyms with Forgeable Identifiers

Carole Delporte-Gallet*, Hugues Fauconnier, and Hung Tran-The

LIAFA- Université Paris-Diderot
{cd,hf,Hung.Tran-The}@liafa.jussieu.fr

Abstract. We consider here the Byzantine Agreement problem (BA) in synchronous systems with *homonyms* in the case where some identifiers may be forgeable. More precisely, the n processes share a set of l ($1 \leq l \leq n$) identifiers. Assuming that at most t processes may be Byzantine and at most k ($t \leq k \leq l$) of these identifiers are forgeable in the sense that any Byzantine process can falsely use them, we prove that Byzantine Agreement problem is solvable if and only if $l > 2t + k$.

Moreover we extend this result to systems with authentication by signatures in which at most k signatures are forgeable and we prove that Byzantine Agreement problem is solvable if and only if $l > t + k$.

1 Introduction

Most of distributed algorithms assume that each process has an unique identity. Yet assuming that every process has unique identity given by a checkable identifier might be too strong (and costly) an assumption in practice especially if some processes are malicious. For example, very simple systems giving MAC addresses as identifier are not reliable, because MAC addresses may be duplicated and more sophisticated mechanisms using for example digital signatures are costly and difficult to implement. Moreover, for privacy reason, the processes may prefer to not have an unique identifier. For example, agents may be registered as members of groups and may want to act as member of the groups not as individuals. Hence it could be useful and interesting to relax the unicity of identifiers assumption. However, lack of identifiers is very restrictive and in fully anonymous systems very few problems are solvable (e.g. [1,3,4,12]).

In [6], the authors presented a general model with homonyms in which processes may share the same identifier and may then be homonyms. In this model n processes share a set of l identifiers ($1 \leq l \leq n$). More precisely, each process p has an unique identifier belonging to the set of l identifiers, but several processes may have the same identifier. The processes cannot distinguish between processes having the same identifier. A process may only send messages to *all the* processes with some identifier and when a process receives a message it knows only the identifier of the origin of the message without knowing which particular process it is. When $l = n$ each process has its own identifier and at the other extreme the case $l = 1$ corresponds to the fully anonymous system.

* This work is supported by the ANR VERSO SHAMAN.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 171–182, 2012.

An important point in the model described in [6] is that there is no "masquerading" and even a Byzantine process is not able to lie about its identifier: if any process p receives a message, then process p knows that the sender of this message has identifier id. Hence even a Byzantine process cannot lie about its identifier. Yet it is rather natural that Byzantine processes may at least form a coalition and "exchange" their identifiers.

In this paper we present an extension of homonym processes in which some identifiers are "forgeable" and a Byzantine process may freely use any such identifier. Moreover we restrict ourselves to the classical synchronized rounds model. More precisely, we keep on ensuring that each process has an unique identity, but some identifiers are forgeable in the sense that Byzantine processes may use such an identifier id to send messages. A process receiving this message falsely believes that the identifier of the sender is id. Of course the set of forgeable identifiers is not known by the processes, we only assume that we have l identifiers for the n processes, and among these identifiers at most k are forgeable. As Byzantine are able to form coalitions and exchange their identifies, we assume that the set of forgeable identifiers contains at least all identifiers of Byzantine processes. Hence if t is the maximum number of Byzantine processes, we have $t \leq k \leq l$.

To determine the power of the model of homonyms with forgeable identifiers, as in [6], we are going to consider the problem of Byzantine Agreement [14]. As a Byzantine process is able to send in each round messages with all forgeable identifiers, intuitively, it means that it is the same as having at least one Byzantine process per forgeable identifier. Recall from [6] that Byzantine Agreement is solvable in the homonyms model if and only if $l > 3t$, then considering forgeable identifiers as similar to groups of processes with Byzantine processes, we get directly a solution with l forgeable identifiers if $l > 3k$ and we could suppose that we have a solution if and only if $l > 3k$. But surprisingly, we prove a better bound, we prove that there is solution for the Byzantine Agreement with k forgeable identifiers if and only if $l > 2t + k$. In fact, this result comes from the fact that if a Byzantine process forges the identifier of a group of processes containing correct processes, this group of processes has the same behavior as a group of processes containing together Byzantine and correct processes. It is proven in [7] that such groups of processes are weaker adversaries than groups containing only Byzantine processes.

From a more practical point of view, it is easy to implement homonyms with help of digital signatures as with [10] in which at each identifier is associated a public key and processes with the same identifiers share corresponding private keys. In this way we get a (strictly) stronger authentication mechanism as defined in [15]. With this authentication mechanism a process cannot retransmit falsely messages. More precisely, if the identifier is unforgeable, then it is not possible for any process q to wrongly pretend that it received message m coming from a process with this identifier. It is well known that with this kind of authentication, the Byzantine Agreement problem can be solved if and only if $n > 2t$ in the classical case in which all processes have unique and different

unforgeable identifiers, giving a $n > 2k$ bound with k forgeable identifiers. With homonyms and at most k forgeable identifiers, we prove that Byzantine Agreement is solvable if and only if $l > t+k$. Again the *a priori* expected result would be $l > 2k$.

Due to the lack of space some proofs are omitted and are in [8].

The rest of the paper is organized as follows. Section 2 describes the model of homonyms with forgeable identifiers and gives the specification of Byzantine Agreement. Then in Section 3 we prove the impossibility results concerning Byzantine Agreement with homonyms and forgeable identifiers. In Section 4, we propose a specification of Authenticated Broadcast and give a corresponding algorithm. Section 5 contains the algorithm for Byzantine Agreement using Authenticated Broadcast. Then, in Section 6 we study the authentication case. Finally in Section 7, we discuss some related work and perspectives.

2 Model and Definitions

Identifiers and homonyms. We consider a distributed message-passing system of n processes. Each process gets an unique identifier from a set of identifiers $\mathcal{L} = \{1, 2, \ldots, l\}$. We assume that each identifier is assigned to at least one process but some processes may share the same identifier. Hence we have $l \leq n$. If p is a process then $Id(p)$ is the *identifier* of process p.[1] For an identifier id, the group of processes with identifier id, $G(id)$, is the set of all processes with identifier id.

For example, if $l = 1$ then the system is fully anonymous and if $n = l$ each process has an unique identifier.

Process failure. A *correct process* does not deviate from its algorithm specification. Some processes may be *Byzantine*, such a process can deviate arbitrarily from its algorithm specification. In particular, a Byzantine process may send different messages than its algorithm specifies or fail to send the messages it is supposed to. In the following t is an upper bound on the number of Byzantine processes.

From [15,14], we know that Byzantine Agreement is impossible to solve if $n \leq 3t$, so we assume $n > 3t$.

Forgeable Identifiers. To each message m is associated the *origin group* of m that is an identifier in $\mathcal{L}$ denoted $from(m)$. When the sender of the message m is a correct process p, this identifier is the identifier of this process: $from(m) = Id(p)$. Remark that $from(m)$ enables only to know the origin group but does not enable to know which process in this group is the sender.

We assume that Byzantine processes have the power to forge some identifiers. The set of identifiers that can be forge by Byzantine processes is a subset of $\mathcal{L}$ and is denoted $\mathcal{F}$. In the following k designs an upper bound of the number of identifiers that can be forged: $|\mathcal{F}| = k$.

[1] For convenience, we sometimes refer to individual processes using names like p but these names cannot be used by processes in the algorithms.

Let id_f be an identifier in $\mathcal{F}$, a Byzantine process with identifier id may send a message m to group id' with the forged identifier id_f. In this case, a process p with identifier id' receives the message m with $from(m) = id_f$. As a Byzantine process acts as an adversary it may divulge any information, then we assume here that if p is a Byzantine process then $Id(p)$ is also in $\mathcal{F}$.

Consequently, if a process p receives a message m with $from(m) = id$, p knows that this message has been either sent by a correct process with identifier id or sent by a Byzantine process which has forged the identifier id.

We name (n, l, k, t)-homonym model such a model. In the following a *correct* group designs a group of processes with some identifier that contains only correct processes and whose its identifier is not forgeable.

Synchronous rounds. We consider a *synchronous model* of rounds. The computation proceeds in rounds. In each round, each process first sends a set of messages, which depends on its current state, to some identifiers. Then, each process receives all the messages sent to its identifier in the same round and finally changes its state according to the set of received messages.

As several processes may share the same identifier, in a round a process may receive several identical messages coming from processes with the same identifier (or Byzantine process that has forged this identifiers). But we assume here that when a process receives a message m with $from(m) = id$, it does not know how many correct processes with identifier id (or Byzantine process that has forged id) have sent this message.[2]

A Byzantine process can deviate arbitrarily from its algorithm specification and Byzantine process may send any set of messages (possibly an empty set) with any identifiers in $\mathcal{F}$. Moreover, contrary to correct processes, we assume that Byzantine processes are able to send different messages to different processes in the same group.

Byzantine Agreement. In the following we are interested in the Byzantine agreement problem [15,14]. Recall that solutions to this problem are the basis of most of fault tolerant algorithms (e.g. [16]). *Byzantine Agreement* is an irrevocable decision problem that has to satisfy the following properties:

1. *Validity:* If all correct processes propose the same value v, then no value different from v can be decided by any correct process.
2. *Agreement:* No two correct processes decide different values.
3. *Termination:* Eventually every correct process decides some value.

3 Impossibility Result

Following the spirit of the impossibility of Byzantine Agreement in [9], we prove our impossibility results in (n, l, k, t)-homonym model.

[2] Our results can be extended to the model of *numerate* processes as defined in [6] for which each process receives in a round a multiset of messages and is able to count the number of copies of identical messages it receives in the round.

Proposition 1. *Byzantine Agreement is unsolvable in* (n, l, k, t)*-homonym model if* $l \leq 2t + k$.

Proof. It suffices to prove there is no synchronous algorithm for Byzantine Agreement when $l = 2t + k$. To derive a contradiction, suppose there is an algorithm $\mathcal{A}$ for Byzantine agreement with $l = 2t + k$. Let $\mathcal{A}_i(v)$ be the algorithm executed by a process with identifier i when it has input value v.

We divide the set of processes into 4 subsets: $A = \cup_{0<i\leq t} G(i)$, $B = \cup_{t<i\leq 2t} G(i)$, $C = \cup_{2t<i\leq 3t} G(i)$ and $F = \cup_{3t<i\leq k+2t} G(i)$.

We consider a system of 8 sets: two sets A_0 and A_1 (resp B_0 and B_1, C_0 and C_1, F_0 and F_1) with the same number of processes and the same repartition in group as A (resp. B, C, F). Each process of A_0 with identifier i executes the code $\mathcal{A}_i(0)$ and the analog for others sets and other input values.

Imagine setting up a system S as shown in Figure 1. Every process correctly executes the algorithm assigned to it. Communication between groups are indicated by continuous line. A dash arrow from group X to group Y indicates that processes of X send their messages also to Y. (We do not pretend that we have a Byzantine Agreement algorithm for this system.)

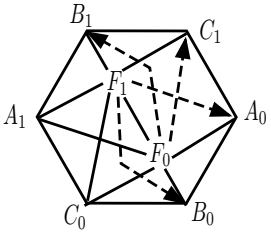


Fig. 1. System S

We now define three executions of the algorithm $\mathcal{A}$.

In execution α, processes of A, B and F are correct and have input 1, all the processes in C are Byzantine and the identifiers in F may be forged. By validity, all the correct processes must decide 1. A process c in C sends (1) the same messages to processes in A and F as the corresponding process in C_0 in system S, (2) the same messages to processes in B and F as the corresponding process in C_1 in system S. Moreover one process in C sends the same messages as processes in F_0 to processes in A, B and F.

In execution β, processes of B, C and F are correct and have input 0, all the processes in A are Byzantine and the identifiers in F may be forged. By validity, all the correct processes must decide 0. A process a in A sends (1) the same messages to processes in C and F as the corresponding process in A_1 in system S, (2) the same messages to processes in B and F as the corresponding process in A_0 in system S. Moreover one process in A sends the same messages as processes in F_1 to processes in A, B and F.

In execution γ, processes of A, C and F are correct. Processes of A and F have input 1 and processes of C have input 0. All the processes in B are Byzantine and

the identifiers in F may be forged. Processes in B send (1) the same messages to processes in A and F as the corresponding processes in B_1 in system S (and the same that in α) (2) the same messages to processes in C as the corresponding processes in B_0 in system S (and the same that in γ).

Moreover one process in B sends the same messages as processes in F_0 to processes in A, C and F. Note that processes in F and this Byzantine process sends the same message to processes in A as in run α and to processes in C as in run γ.

So, for the correct processes in A executions α and γ are undistinguishable and they decide 1. For the correct processes in C executions β and γ are undistinguishable and they decide 0. These decisions in execution γ give the contradiction.

4 Authenticated Broadcast

Our algorithms for Byzantine Agreement use an adaptation of Authenticated Broadcast as introduced by Srikanth and Toueg [17] in the classical case where each process has a different identifier ($n = l$).

First, recall some principles for Authenticated Broadcast. In [17], it is defined by two primitives: $Broadcast(p, m, r)$ and $Accept(p, m, r)$. Computation is broken up in *superrounds*. Roughly speaking a process p broadcasts a message m in superround r by $Broadcast(p, m, r)$. If the process p is correct all processes receive the message and $Accept(p, m, r)$. If the process p is a Byzantine process, the Authenticated Broadcast guarantees that if some process $Accept(p, m, r)$ at some superround r' then all correct processes $Accept(p, m, r)$ no later than in superround $r' + 1$. Furthermore no correct process can $Accept(p, m, r)$ from a correct process p if p has not broadcast it.

Considering the (n, l, k, t)-homonym model, we decompose the synchronous computation in superrounds too. All the processes of a group have to invoke broadcast of a message m in the superround r in order to ensure that this message will be accepted in the following superround.

More precisely, our Authenticated Broadcast is defined by two primitives: $Broadcast(i, m, r)$ and $Accept(i, m, r)$ where i is the identifier of some group. We assume that a correct process broadcasts at most one message in a superround. The Authenticated Broadcast primitive is specified as follows:

1. *Correctness:* If all the processes in a correct group i perform $Broadcast(i, m, r)$ in superround r then every correct process performs $Accept(i, m, r)$ during superround r.
2. *Relay:* If a correct process performs $Accept(i, m, r)$ during superround $r' \geq r$ then every correct process performs $Accept(i, m, r)$ by superround $r' + 1$.
3. *Unforgeability:* If some correct process performs $Accepts(i, m, r)$ in superround $r' \geq r$ then all correct processes in group i must $Broadcast(i, m, r)$ in superround r.

The algorithm is described in Figure 2. A superround r is composed of the two rounds $2r$ and $2r + 1$.

Code for process p with identifier $i \in \{1, ..., l\}$

Variable:

1 $\mathcal{M} = \emptyset$;

Main code:

2 ROUND R
3 **if** $R = 2r$ **then if** $Broadcast(i, m, r)$ **to perform**
4 **then send** $(\mathcal{M} \cup (init, i, m, r), R)$ to all
5 **else send** $(\mathcal{M} \cup (noinit, i, \perp, r), R)$ to all
6 **else send** $(\mathcal{M}, R)$ to all;

Reception of the messages of round R

7 **For all** $h \in \{1, ..., l\}$
8 **if** $(R = 2r)$ **then**
Let $\mathcal{M}[h]$ be the set of messages $(init, h, *, r)$ or $(noinit, h, *, r)$ received from processes in group h
9 **if** $M[h] = \{(init, h, m, r)\}$
10 **then** $\mathcal{M} = \mathcal{M} \cup (echo, h, m, r)$
11 **For all** $r \in \{1, ..., R/2\}$
12 **For all** $m \in$ possible messages
13 **if** $(echo, h, m, r)$ received from at least $l - 2t$ distinct groups
14 **then** $\mathcal{M} = \mathcal{M} \cup (echo, h, m, r)$
15 **if** $(echo, h, m, r)$ received from at least $l - t$ distinct groups
16 **then** $Accept(h, m, r)$

Fig. 2. Authenticated Broadcast algorithm in the (n, l, k, t)-homonym model

First, recall the principles of the algorithm of [17] with T Byzantine processes. To propose a value v in superround r, process p sends message $(init, p, v, r)$ to all processes (including itself). A process receiving such a message becomes a "witness" for (p, v, r) and sends a message of type *echo* to all processes. Any process that has $T + 1$ witnesses for (p, v, r) becomes itself witness (because at least one correct process has sent this message). When a process receives more than $(2T + 1)$ witnesses, it accepts (p, v, r) ($T + 1$ correct processes have sent this message then all correct processes will find $T + 1$ witnesses of this message).

The algorithm follows the same principles, to propose a value v in supperround r process p with identifier i sends message $(init, i, v, r)$ to all processes (including itself) (line 4). A process receiving such a message from some processes with identifier i becomes "witness" for (i, v, r) and sends a message of type *echo* to all processes (line 10). Any process having $l - 2t$ witnesses for (i, v, r) becomes itself witness (if $l - 2t > k$ at least one process in a correct group has sent this message) (line 13). When a process receives more than $(l - t)$ witnesses, it accepts (i, v, r) (at least $t + 1$ processes from correct groups have sent this message) (lines 15 to 16). In this way we ensure *correctness* and *relay* properties.

To ensure the unforgeability property, a correct process that has no message to broadcast in a superround broadcast a *noinit* message in the corresponding even round. In this way, if some correct process with identifier i has no message to broadcast in superround r, every correct process gets $M[i]$ (line 8) different from one message *init* and will not become witness of any message $(i, *, r)$.

By a standard proof, we get :

Proposition 2. *If $l > 2t + k$, the algorithm Figure 2 implements Authenticated Broadcast in (n, l, k, t)-homonym model.*

5 Byzantine Agreement Algorithms

Our algorithm follows the line of the algorithm in [17] defined in the classical case where each process has a different identifier ($n = l$). One of the main difference here is the fact that the behavior of groups of processes is different from the behavior of processes. In particular, correct processes in groups with forgeable identifiers or in groups containing Byzantine processes have to decide and have to participate to the decision.

The algorithm proceeds in synchronous superrounds (set of successive rounds). In this algorithm, 1 is the only value that may be broadcast and value 0 is decided upon by default if 1 is not decided. Hence, all processes in superround 1 broadcast 1 if their input is 1. A process p sets variable *value* to 1 (and then will decides 1) in superround $r < 2k$, if it has (1) accepted $(i_u, 1, 1)$ messages from $t + 1$ distinct identifiers i_u, and (2) accepted $(j_u, 1, r_u)$ from $(r + 1)/2$ with $r_u \geq 2$. Condition (1) ensures that at least one correct group has broadcast and condition (2) corresponds to the one of [17]. Condition (1) ensures the *validity* property of consensus and the condition (2) ensures the *agreement* property. The *correctness* and *relay* properties of Authenticated Broadcast ensure that by the next superround (superround $r + 1$), all messages accepted by process p are also accepted by all correct processes. Hence, they all set the variable *state* to *true*. By superround $r + 2$, they broadcast and by *correctness* and *relay* properties, all correct processes decide 1 by setting *value* to 1. If p decides in superround $2k + 2$, then it can be proved that at least one group of correct processes set its variable *value* to 1 by superround $2k$, and all correct processes decide by superround $2k + 2$.

We now present the steps of the proof that the algorithm of Figure 3 satisfies the specification of Byzantine agreement.

Lemma 1. *If, at superround r, a correct process p has $|A_r| \geq t+1$ then at each superround r' with $r + 1 \leq r' \leq 2k + 2$ each correct process has $|A_{r'}| \geq t + 1$.*

Lemma 2. *If every process of a correct group broadcasts some message in some superround $r > 1$, then every correct process sets value to 1 by superround r.*

Proposition 3. *(Validity) If all correct processes propose the same initial value v then no value different from v can be decided by any correct process.*

Code for process p with identifier i

Variable:

1 $input = \{v\}$ /* $v \in \{0, 1\}$ is the value proposed value */
2 $value = 0$
3 $state$= false

Main code:

4 SUPERROUND 1
5 **if** $input = 1$ **then** $Broadcast(i, 1, 1)$
Let $A_1 = \{h : p$ has $Accepted(h, 1, 1)\}$
6 **if** $|A_1| \geq t + 1$ **then** $state = true$
7 SUPERROUND r **from** 2 **to** $2k + 2$
8 **if** $state = true$ **then** $Broadcast(i, 1, r)$; $state = false$
Let $A_r = \{h : p$ has $Accepted(h, 1, 1)\}$
9 **if** $|A_r| \geq t + 1$ and
has $Accepted(i_u, 1, r_u)$ from $\frac{r+1}{2}$ distinct identifiers i_u with $r_u \geq 2$
10 **then** $value = 1$
11 **if** $|A_r| \geq t + 1$ and
has $Accepted(i_u, 1, r_u)$ from $\frac{r}{2}$ distinct identifiers i_u ($i_u \neq i$) with $r_u \geq 2$
12 **then** $state = true$;
13 AT THE END OF SUPERROUND $2k + 2$
14 **if** $value = 1$
15 **then** DECIDE 1
16 **else** DECIDE 0

Fig. 3. Synchronous Byzantine Agreement algorithm in (n, l, k, t)-homonym model

Proposition 4. *(Termination) Eventually every correct process decides some value.*

Assume that, in the execution, some correct process sets *value* to 1 at Line 10, and decides 1. Let r_1 be the first superround in which some correct process sets *value* to 1. Let p_1 be such a correct process that sets *value* to 1.

With the help of Lemma 2, we prove:

Lemma 3. *If $r_1 \leq 2k$, then every correct process sets value to 1 by superround $r_1 + 2$.*

When r_1 is greater than $2k$, p_1 has accepted $(j_u, 1, r_u)$ from $\frac{2k+2}{2}$ distinct processes j_u, then from at least $k + 1$ distinct processes. Then there is at least one correct group in this set. Then we get:

Lemma 4. *If $r_1 > 2k$, then every correct process sets value to 1 by superround r_1.*

Proposition 5. *(Agreement) No two correct processes decide different values.*

From Propositions 3, 4 and 5, we have:

Theorem 1. *Assuming if $l > t+k$, algorithm of Figure 3 implements Byzantine Agreement using Authenticated Broadcast in (n, l, k, t)-homonym model.*

Combining this with algorithm for Authenticated Broadcast, we get:

Corollary 1. *If $l > 2t+k$, the algorithms of Figure 2 and 3 implement Byzantine Agreement in (n, l, k, t)-homonym model.*

6 Authentication

Authentication [15] ensures that Byzantine process may "lie" about its own values but may not relay altered values without betraying itself as faulty. Currently this property may be ensured with a system of signatures using public keys cryptography.

The implementation of homonyms with authentication is rather natural: the members of a group share a secret key. The origin group of a message may be verified by a signature scheme of this group. Messages m are authenticated by the identifier of the sender. Each process can verify if a message carries the signature of a given identifier. As before we assume that the signatures of at most k identifiers can be forged.

With homonyms and this scheme of authentication, if id is not forgeable (any process with this identifier is correct) then it is not possible for any process to wrongly pretend that it has received some messages coming from identifier id, hence we get the authentication property of [15].

For this model with authentication, we improve our bound: $l > t + k$ is necessary and sufficient to achieve Byzantine Agreement. (Recall that in the classical model in which each process has its own identifier and without forgeable identifiers the bound is $n > 2t$.)

The proofs of the lower bound is essentially the same as for the case without authentication in Section 3.

The Byzantine Agreement algorithm of Figure 3 directly works with $l > t + k$ if we have an Authenticated Broadcast. It remains to get an Authenticated Broadcast with $l > k + t$.

In [17], in the classical case in which each process has its own identity, Authenticated Broadcast can be obtained simply with authentication: it suffices to verify the signatures. A process that receives a message (p, m, r) accepts it if it can verify p's signature (and then forwards this message). But if we apply this simple mechanism in our model, we do not get the unforgeability property. In a forgeable group with identifier i containing some correct processes, it is possible that some correct process accepts (i, m, r). Indeed, this message has been sent by a Byzantine process that has forged the identifier i.

To implement Authenticated Broadcast, we use the signatures and the mechanism of witnesses as in our previous algorithm. The implementation is based on the one presented in Section 4 with some simple changes.

At some point, in algorithm 2 when a process receives some messages in round R, it will send *echo* in the next rounds. The process will forward all messages

that produced this *echo*. In this way it gives a proof that it has the right to send *echo*. We get an Authenticated Broadcast algorithm from algorithm Figure2 by: (1) removing from the received message all messages with a bad signature, (2) forwarding the proof of each new *echo* message, (3) replacing the line 13 of algorithm 2 by:

if received $(echo, h, m, r)$ and the proof of $(echo, h, m, r)$

and, (4) replacing the line 15 of algorithm 2 by

if received $(echo, h, m, r)$ and the proof of $(echo, h, m, r)$ from at least $l - t$ distinct groups

We have:

Theorem 2. *With authentication, Byzantine Agreement is solvable in* (n, l, k, t)*-homonym model if and only if* $l > t + k$.

7 Related Works and Perspectives

When processes share identifiers and some of theses processes may be Byzantine, it is rather natural that some of these identifiers may be forged. Hence this work is a natural extension of [6] to forgeable identifiers.

At least for the authentication case, groups signature as introduced first in [5] are close to our model. Groups signature enable to sign messages on behalf of a group and clearly can be used to implement the model of homonyms. Note that group signatures generally ensures other properties than the one we consider here. Groups signatures may be a valuable way to implement models with homonyms.

In other works [2,11,13], a mixed adversary model is considered in the classical ($l = n$): the adversary can corrupt processes actively (corresponding to Byzantine process) and can forge the signature of some processes.

In some way, we combine here the idea of group signatures and forgeable signatures but contrary to group signatures the goal is not to develop protocol ensuring strong properties like anonymity or unforgeability but try to develop algorithms (like agreement) in presence of groups of processes with Byzantine processes and forgeable identifiers.

Here we proved that with forgeable identifiers and homonyms, Byzantine Agreement can be solved in a "reasonable" way (and without any assumption about cryptographic system). Interestingly, the solvability of Byzantine Agreement depends only on the number of identifiers and the number of forgeable identifiers. Hence adding correct processes does not help to solve Byzantine Agreement.

A natural extension of this work could be to consider partially synchronous models.

As Byzantine Agreement is the basis for replication systems in the classical models in which each process has its own identity, a natural question is to know if it is still the case and envisage to develop algorithms for more difficult problems.

References

1. Attiya, H., Gorbach, A., Moran, S.: Computing in totally anonymous asynchronous shared memory systems. Information and Computation 173(2), 162–183 (2002)
2. Bansal, P., Gopal, P., Gupta, A., Srinathan, K., Vasishta, P.K.: Byzantine Agreement Using Partial Authentication. In: Peleg, D. (ed.) DISC 2011. LNCS, vol. 6950, pp. 389–403. Springer, Heidelberg (2011)
3. Boldi, P., Vigna, S.: An Effective Characterization of Computability in Anonymous Networks. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 33–47. Springer, Heidelberg (2001)
4. Buhrman, H., Panconesi, A., Silvestri, R., Vitányi, P.M.B.: On the importance of having an identity or, is consensus really universal? Distributed Computing 18(3), 167–176 (2006)
5. Chaum, D., van Heyst, E.: Group Signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991)
6. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Kermarrec, A.-M., Ruppert, E., Tran-The, H.: Byzantine agreement with homonyms. In: PODC, pp. 21–30. ACM (2011)
7. Delporte-Gallet, C., Fauconnier, H., Tran-The, H.: Byzantine Agreement with Homonyms in Synchronous Systems. In: Bononi, L., Datta, A.K., Devismes, S., Misra, A. (eds.) ICDCN 2012. LNCS, vol. 7129, pp. 76–90. Springer, Heidelberg (2012)
8. Delporte-Gallet, C., Fauconnier, H., Tran-The, H.: Homonyms with forgeable identifiers. Technical Report hal-00687836, HAL-CNRS (April 2012)
9. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. Distributed Computing 1(1), 26–39 (1986)
10. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
11. Gordon, S.D., Katz, J., Kumaresan, R., Yerukhimovich, A.: Authenticated Broadcast with a Partially Compromised Public-Key Infrastructure. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 144–158. Springer, Heidelberg (2010)
12. Guerraoui, R., Ruppert, E.: Anonymous and fault-tolerant shared-memory computing. Distributed Computing 20(3), 165–177 (2007)
13. Gupta, A., Gopal, P., Bansal, P., Srinathan, K.: Authenticated Byzantine Generals in Dual Failure Model. In: Kant, K., Pemmaraju, S.V., Sivalingam, K.M., Wu, J. (eds.) ICDCN 2010. LNCS, vol. 5935, pp. 79–91. Springer, Heidelberg (2010)
14. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Transactions on Programming Languages and Systems 4(3), 382–401 (1982)
15. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (1980)
16. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys 22(4), 299–319 (1990)
17. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distributed Computing 2(2), 80–94 (1987)

Asynchrony and Collusion in the N-party BAR Transfer Problem

Xavier Vilaça, Oksana Denysyuk, and Luís Rodrigues

INESC-ID, Instituto Superior Técnico, Universidade Técnica de Lisboa

Abstract. The problem of reliably transferring data from a set of $N_{\mathcal{P}}$ producers to a set of $N_{\mathcal{C}}$ consumers in the BAR model, named N-party BAR Transfer (NBART), is an important building block for volunteer computing systems. An algorithm to solve this problem in synchronous systems, which provides a Nash equilibrium, has been presented in previous work. In this paper, we propose an NBART algorithm for asynchronous systems. Furthermore, we also address the possibility of collusion among the Rational processes. Our game theoretic analysis shows that the proposed algorithm tolerates certain degree of arbitrary collusion, while still fulfilling the NBART properties.

1 Introduction

Peer-to-peer networks can be used for executing computationally intensive projects, as shown by the Boinc infrastructure [1]. Building systems on this kind of networks may be quite challenging due to the existence of Byzantine processes, whose behaviour is arbitrary, and of Rational processes, which may deviate from the specified protocols if they can increase their utility. A system model that captures this variety of behaviours has been coined the BAR model [2], named after the three classes of processes (Byzantine, Altruistic, and Rational) that it explicitly considers.

Our work focuses on the particular problem of reliably transferring data from a set of $N_{\mathcal{P}}$ producers to a set of $N_{\mathcal{C}}$ consumers in the BAR model, named N-party BAR Transfer (NBART). Although an algorithm that solves this problem has already been devised for synchronous systems [3], in a peer-to-peer network it is often unrealistic to assume that there is a known upper bound for the execution time and the communication delay. With this in mind, this paper addresses the NBART problem in an asynchronous system. Furthermore, this paper also addresses the problem of collusion, which is a real issue in peer-to-peer networks due to attacks, such as sybil and white washing. In addition to arbitrary collusion of Byzantine players, we consider that Rational processes may create collusion groups, including producers and consumers.

Related Work. Models based on traditional Game Theory assume that all processes are Rational, and they fail to account for arbitrary behaviour that may arise from Byzantine faults. To the best of our knowledge, the work of Eliaz

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 183–194, 2012.

et. al. [4] was the first to accommodate Byzantine-awareness, by introducing the notion of k-Fault Tolerant Nash Equilibrium (k-FTNE). In this context, a profile of strategies is k-FTNE if the strategy of each player is a best response to the strategy of other players, independently of the identity of Byzantine players and the arbitrary strategy they follow. This concept was later applied to virus inoculation games [5].

Aumann [6] addressed the issue of collusion by defining an equilibrium as a profile of strategies where no deviating collusion strategy provides a greater utility for all players of the group. Then, Bernheim et. al. [7] introduced the notion of *coalition-proof Nash equilibrium*, where no deviations by a coalition can perform better, although they do not allow further deviations to the collusion strategy. This work was later extended to take into consideration correlated strategies [8].

The work of [9] considered the existence of processes with unexpected utilities and collusion. The authors proposed the solution concept of (k, t)-robustness, where no process can increase its utility by deviating in collusion with up to $k-1$ other processes, regardless of the Byzantine behaviour of up to t processes. This notion is stronger than the previous models for collusion, since it accounts for arbitrary collusion where it should be true that no player performs better by deviating from the equilibrium strategy, even if that implies decreasing the utility of other players within the coalition. Unfortunately, in certain scenarios such as communication games (where players incur communication costs), it was shown that no game can be (k, t)-robust for $k, t > 0$ [10].

Additional literature relevant to our results include works on agreement in the BAR model [2,10] and data dissemination [11,12,13], which studied protocols tolerant to the BAR model. In [14], the authors studied the impact of altruism on a repeated game modelled by the BAR model. All these works assume repeated interactions of processes in a cooperative service. On the other hand, our paper considers one-shot games, and therefore addresses the need to provide equilibrium strategies for Rational processes to follow the specified algorithm based on incentives provided in a single instance of NBART.

Contributions. The first contribution of this paper consists in an algorithm that solves NBART in asynchronous systems. We show that the proposed algorithm is correct, assuming that all non-Byzantine processes follow it, for $N_{\mathcal{P}} \geq 2F_{\mathcal{P}}+1$ and $N_{\mathcal{C}} \geq F_{\mathcal{C}}+1$, where $F_{\mathcal{P}}$ and $F_{\mathcal{C}}$ are upper bounds on the number of Byzantine producers and Byzantine consumers respectively. We also show that the presented algorithm obtains asymptotically optimal bit complexity in certain scenarios.

The second contribution consists in the game theoretic analysis of the proposed algorithm. Since processes incur communication costs, our algorithm cannot be (k, t)-robust [10], hence we rely on a weaker notion of Byzantine aware utility function to account for Byzantine behaviour, based on the notion proposed in [10]. Given that we cannot ensure that the players within a coalition follow the algorithm, we propose a new solution concept, which is an adaptation of k-resilience to account for collusion in the following way. We define an

equilibrium as a profile of strategies $\boldsymbol{\sigma}$ where members of a coalition are interested in deviating from $\boldsymbol{\sigma}$ only if their behaviour, as observed by other processes, is equivalent to $\boldsymbol{\sigma}$. We assume that the size of each group of Rational colluding processes is bounded by a constant $N_{\mathcal{T}} = N_{\mathcal{T}}^{\mathcal{P}} + N_{\mathcal{T}}^{\mathcal{C}}$, where $N_{\mathcal{T}}^{\mathcal{P}}$ is the number of members of the colluding group that are producers and $N_{\mathcal{T}}^{\mathcal{C}}$ is the number of consumers on the same group. We show that, if $N_{\mathcal{P}} \geq \max(F_{\mathcal{P}}, N_{\mathcal{T}}^{\mathcal{P}}) + F_{\mathcal{P}} + 1$ and $N_{\mathcal{C}} \geq F_{\mathcal{C}} + N_{\mathcal{T}}^{\mathcal{C}} + 1$, then the algorithm provides such equilibrium, implying that processes from any coalition follow a strategy that ensures that the NBART properties are fulfilled. An important consequence of this is that, in the absence of collusion, the algorithm provides a Nash equilibrium.

2 System Model

We assume an asynchronous system composed of N *processes* or *players* (we will use the term *player* only when performing the Game Theoretic analysis; in any other case, we will use the name process). Processes are connected by a fully-connected network and can communicate using reliable authenticated point-to-point communication channels [15].

We make the distinction between *identity*, *process/player*, and *coalition*. An identity is a tuple (i, pk_i, sk_i), where i is an identifier and pk_i and sk_i are the corresponding public and private keys. There is a set of identities $\mathcal{I} = \mathcal{P} \cup \mathcal{C}$, where $\mathcal{P}$ and $\mathcal{C}$ are the sets of producer and consumer identities, respectively, such that $\#\mathcal{P} = N_{\mathcal{P}}$ and $\#\mathcal{C} = N_{\mathcal{C}}$. Players are the decision-making entities of our Game Theoretic analysis and are represented by a single identity. Therefore, when referring to the process that holds the identity (i, pk_i, sk_i), we will simply refer to it as i. If $i \in \mathcal{P}$, the corresponding process is referred to as a producer, otherwise, it is called a consumer. Finally, $N_{\mathcal{P}} + N_{\mathcal{C}} = N$.

As defined by the BAR model, a player can be Altruistic (if it follows the algorithm), Byzantine (if its behaviour is arbitrary), or Rational (if it follows the strategy that maximises its utility given the expectations regarding the strategies followed by other players). We assume that Rational processes adhere to the *promptness principle* [2], in the sense that if the expected utilities of following the algorithm and deviating by delaying messages are equivalent, then processes do not deviate. It is said that a player i signs information with sk_i by invoking $s_i(data)$.

2.1 NBART Problem

The NBART Problem can be defined as follows. Each producer p produces an arbitrarily large value v_p by invoking the deterministic function *produce*(p, v_p), such that any two non-Byzantine producers produce the same value, named the *correct value*. Consumers must consume only one value v, sent by some producer, by invoking *consume*(c, v). The invocation of this primitive proves that, indeed, c consumes the value. To deal with Rational behaviour, we rely on the participation of an abstract entity named Trusted Observer (TO), whose

function is to gather cryptographic information from the participants of each transfer and reward processes according to their observable behaviour. To assess the behaviour of each process, TO uses two predicates *hasProd*(evidence, p) and *hasAck*(evidence, c) that take as input the evidence produced by *TO* to indicate, respectively, if producer p participated in NBART and if consumer c notified the reception of the correct value. *TO* is said to eventually *produce evidence* about the transfer if it invokes *certify*(*TO*, *evidence*) after the moment when *hasProd* and *hasAck* become true for all corresponding non-Byzantine producers and consumers. With these definitions, the NBART problem is characterised by the following properties:

- **NBART 1** (*Validity*): If a non-Byzantine consumer consumes v, then v was produced by some non-Byzantine producer.
- **NBART 2** (*Integrity*): No non-Byzantine consumer consumes more than once.
- **NBART 3** (*Agreement*): No two non-Byzantine consumers consume different values.
- **NBART 4** (*Eventual Consumption*): Eventually, every non-Byzantine consumer consumes a value.
- **NBART 5** (*Evidence*): TO eventually produces evidence about the transfer.
- **NBART 6** (*Producer Certification*): If producer p is non-Byzantine, then *hasProd(evidence, p)* eventually becomes *true*.
- **NBART 7** (*Consumer Certification*): If consumer c is non-Byzantine, then *hasAck(evidence, c)* eventually becomes *true*.

3 Asynchronous NBART

We now describe an algorithm that solves the NBART problem in an asynchronous environment. We first provide an overview and then proceed to the detailed description of the algorithm.

3.1 Overview of the Algorithm

The algorithm can be briefly described as follows. Each producer p owns a block (b_p) that belongs to the set of $N_{\mathcal{P}}$ blocks obtained from the value v by using Reed-Solomon codes, such that v can be retrieved from any subset of B blocks ($N_{\mathcal{P}} \geq B + F_{\mathcal{P}}$). Then, p strives to transfer b_p along with the signature of the vector that contains the hashes of all blocks to a subset of consumers denoted by $conset_p$. Each consumer c only needs to receive B correct blocks and $F_{\mathcal{P}} + 1$ signatures of the same vector of hashes to consume the value. However, c must continue to process any received information and send it to TO, which must (re-)invoke *certify*(*evidence*) whenever it receives new information, in order to fulfil the property NBART-5.

3.2 Algorithm in Depth

The algorithm is depicted for producers in Alg. 1, for consumers in Alg. 2 and Alg. 3, and for TO in Alg. 4. Producers use Reed-Solomon codes to reduce the communication costs of transferring an arbitrarily large value. The value v, whose length in bits is denoted by l_v, is split into $N_\mathcal{P}$ blocks of size $\frac{l_v}{B}$, such that any subset of B blocks is sufficient to retrieve the original value, where $1 \leq B \leq N_\mathcal{P} - F_\mathcal{P}$ and $B < l_v$. There is a function *RS-ENC*$(v, N_\mathcal{P}, B, \omega)$ that, given the correct value v, the number of producers $N_\mathcal{P}$, the number of blocks B, and the word size ω, returns a vector $\boldsymbol{v}$ containing the $N_\mathcal{P}$ blocks, where $2^\omega > N_\mathcal{P}$. Let $\boldsymbol{h}_v$ denote the vector containing the hashes of each of the blocks from $\boldsymbol{v}$. The inverse function *RS-DEC*$(\boldsymbol{v'}, N_\mathcal{P}, B, \omega, \boldsymbol{h}_v)$ is defined as follows: if there are at least B blocks from $\boldsymbol{v'}$ whose hash is in $\boldsymbol{h}_v$, then it returns the value v; otherwise, it returns $\perp$. We consider that all arithmetic operations are performed over elements of the Galois Field $GF(2^\omega)$.

We consider that each process is unequivocally identified by an index, between 0 and $N_\mathcal{P} - 1$ for producers, and between 0 and $N_\mathcal{C} - 1$ for consumers. Each consumer c_j uses a deterministic function $prodset_{c_j}$ to determine the set of producers that are supposed to send it their blocks, defined in such a way that each consumer is related to exactly $B+F_\mathcal{P}$ producers (in this way distributing load among producers). A possible mapping function is the following: $prodset_{c_j} = \{p_i \in \mathcal{P} | i \in [k...(k+B+F_\mathcal{P}-1)\ mod\ N_\mathcal{P}], k = j(B+F_\mathcal{P})\ mod\ N_\mathcal{P}\}$. It is useful to define the function that establishes the inverse relation $conset_{p_i} = \{c_j \in \mathcal{C} | p_i \in prodset_{c_j}\}$ for each producer p_i. These definitions ensure that each consumer is able to receive at least B blocks from non-Byzantine producers, therefore being able to retrieve the correct value. In addition, the load is distributed across the producers such that $\forall_{p \in \mathcal{P}} : \#conset_p = n \Rightarrow \forall_{p' \in \mathcal{P} \setminus \{p\}} : n - 1 \leq \#conset_{p'} \leq n + 1$.

Each producer p starts by storing the set of blocks from $\boldsymbol{v}$ by invoking *RS-ENC*. Note that each producer will only be required to transmit one of these blocks (each producer transmits a different block). However, each producer is still required to send $\boldsymbol{h}_v$. Therefore, each producer then sets the vector *hashes* to $\boldsymbol{h}_v$ (Alg. 1, lines 4-7). Then, p transfers its block along with $\boldsymbol{h}_v$ to all consumers of $conset_p$ in a BLOCK message (lines 8-10), while sending SUMMARY messages to the remaining consumers only containing $\boldsymbol{h}_v$ (lines 11-13). Both these messages are signed with the public key of the producer. Notice that, in the BLOCK message, it is not necessary to sign the block, for the signature of the hashes already authenticates the block.

In turn, each consumer c keeps all the received data blocks in a vector *blocks* and the received vectors of hashes (along with the signatures) in *hashvecs*. In addition, there is a set *missing* that keeps the identities of the producers that have not yet sent any signed information. Finally, *correcthashvec* is the correct vector of hashes, that is, the vector that is sent by at least $F_\mathcal{P} + 1$ producers, and *correctproducers* stores, for each producer, the value $\perp$ if it has not yet sent any message, or the signature of the message sent by the producer.

Each consumer uses the functions *verifysig*(*i*,*d*) and *verifyhash*(*b*,*h*) to verify the signature by i of d and the hash of b when compared to h, respectively.

Algorithm 1. NBART ($p \in \mathcal{P}$)

```
01 upon init() do
02    blocks := [⊥]^{N_P};
03    hashes :=[⊥]^{N_P};

04 upon produce(p, v) do
05    blocks := RS-ENC(value,N_P,B,ω);
06    forall i ∈ P do
07       hashes[i] := hash(blocks[i]);
08    signature := s_p(BLOCK||hashes);
09    forall c ∈ conset_p do
10       send(p, c, [BLOCK, blocks[p], hashes, signature]);
11    signature := s_p(SUMMARY||hashes);
12    forall c ∈ C \ conset_p do
13       send(p, c, [SUMMARY, hashes, signature]);
```

Consumer c is in one of three states: *init*, *gotHashes*, and *consumed*. c is in state *init* when *hashvecs* does not contain a majority ($F_{\mathcal{P}} + 1$) of identical vectors of hashes. The function *minimumHashes* (Alg. 2, lines 8-12) marks the transition between *init* and *gotHashes*, by setting *correcthashvec* to a non-null value, when the required majority of hashes is gathered by c. Procedure *consume-and-report* (lines 16-23) makes the transition from *gotHashes* to *consumed* when the consumer gathers at least B correct blocks and, therefore, the invocation of *RS-DEC* returns a non-null value. In this case, the consumer consumes the value (line 19) and prepares a report intended to TO (lines 20-23), which is sent by invoking the procedure *report* (lines 13-15). This report contains the vector *correcthashvec* and the signature of all the producers that already sent correct messages to c, i.e., messages that contained *correcthashvec*.

Whenever a consumer c receives a BLOCK message from a producer that belongs to $missing \cap prodset_c$ (Alg. 3, line 1), c removes p from *missing* if the signature is valid (lines 2-3) and, according to its state, performs one of the following actions: i) If c is still in state *init*, then it stores the received information in the appropriate vectors and invokes *minimumHashes* (lines 4-8), in order to verify if it has already gathered a majority of identical vectors of hashes. If that is the case, then c invokes *consume-and-report* (lines 9-10). ii) If c is in state *gotHashes*, then it adds the received vector of hashes along with the signature to *hashvecs*, stores the block, and invokes *consume-and-report* (lines 11-15). iii) If c is in state *consumed*, then it adds the signature of the producer to *correctproducers* and reports the information received from producers to TO (lines 16-18). An almost identical approach is followed by c whenever it receives a SUMMARY message, aside from the fact that in this case c does not expect to receive any block (lines 19-33).

The trusted observer only waits for REPORT messages from consumers to include all the received information in the array *evidence* (lines 3-5). In addition, TO repeatedly tries to produce the evidence about the transfer whenever it receives new information (line 6).

Algorithm 2. NBART ($c \in \mathcal{C}$): Part I

```
01 upon init do
02   value :=⊥;
03   correcthashvec := ⊥;
04   hashvecs :=[⊥]^{N_P};
05   blocks := [⊥]^{N_P};
06   missing := P;
07   correctproducers := [⊥]^{N_P};

08 function minimumHashes(hashvecs) is
09   if ∃h : #{p|hashvecs[p] = ⟨h, ∗⟩} ≥ F_P + 1 then
10     return h;
11   else
12     return ⊥;

13 procedure report is
14   signature := s_c(REPORT||correcthashvec||correctproducers);
15   send(c, TO, [REPORT, correcthashvec, correctproducers, signature]);

16 procedure consume-and-report is
17   value := RS-DEC(blocks, N_P, B, ω, correcthashvec);
18   if value ≠ ⊥ then
19     consume(c, value);
20     forall p ∈ P do
21       if hashvecs[p] = ⟨correcthashvec,signature⟩ then
22         correctproducers[p] := signature;
23     report ();
```

We now define the predicates *hasProd* and *hasAck*. It is said that producer p *is certified* by consumer $c \in conset_p$ iff evidence$[c] = \langle \boldsymbol{h}_v, \text{report} \rangle$ and report$[p] = s_p(\text{BLOCK}, \boldsymbol{h}_v)$. We say that producer p is certified by consumer $c \in \mathcal{C} \setminus conset_p$ iff evidence$[c] = \langle \boldsymbol{h}_v, \text{report} \rangle$ and report$[p] = s_p(\text{SUMMARY}, \boldsymbol{h}_v)$. Let $\bar{\mathcal{P}} \subseteq \mathcal{P}$ and $\bar{\mathcal{C}} \subseteq \mathcal{C}$ be the greatest sets that fulfil the following conditions: i) for each $p \in \bar{\mathcal{P}}$ and $c \in \bar{\mathcal{C}}$, p is certified by c; and ii) for each $c \in \bar{\mathcal{C}}$, c invokes *consume*(c,v). With this in mind, we now define the predicates as follows. For the predicates to be true for any process, $\#\bar{\mathcal{P}} \geq N_{\mathcal{P}} - F_{\mathcal{P}}$ and $\#\bar{\mathcal{C}} \geq N_{\mathcal{C}} - F_{\mathcal{C}}$. Given this, *hasProd*(evidence,p) is true iff $p \in \bar{\mathcal{P}}$ and *hasAck*(evidence,c) is true iff $c \in \bar{\mathcal{C}}$.

Due to space constraints, we do not include the proofs of correctness and the complexity analysis in this document. We also leave out the proofs of the lemmas and theorems included in the following section. A full version of the paper is available in [16].

4 Game Theoretic Analysis

The purpose of this analysis is to show that it is in every Rational process interest to follow the algorithm. We take into consideration some degree of arbitrary collusion.

4.1 Definitions

The algorithm is modelled as a coalitional game $\Gamma = (\mathcal{I}, \mathcal{T}, \Sigma_{\mathcal{I}}, (\succeq_t)_{t \in \mathcal{T}}, (u_i)_{i \in \mathcal{I}})$. Here, $\mathcal{I} = \mathcal{P} \cup \mathcal{C} \cup \{\text{TO}\}$ is the set of players. $\mathcal{T}$ is the set of non-empty subsets

Algorithm 3. NBART ($c \in \mathcal{C}$): Part II

```
01 upon deliver(p, c, [BLOCK, pblock, phashes, msgsig]) ∧ p ∈ missing ∩ prodset_c do
02   if verifysig(p, BLOCK||phashes, msgsig) then
03     missing := missing \ {p};
04     if verifyhash(pblock, phashes[p]) then
05       if correcthashvec = ⊥ then
06         hashvecs[p] := ⟨phashes, msgsig⟩;
07         blocks[p] := pblock;
08         correcthashvec := minimumHashes(hashvecs);
09         if correcthashvec ≠ ⊥ then
10           consume-and-report ();
11       else if value = ⊥ then
12         if phashes = correcthashvec then
13           hashvecs[p] := ⟨phashes, msgsig⟩;
14           blocks[p] := pblock;
15           consume-and-report ();
16       else if phashes = correcthashvec then
17         correctproducers[p] := msgsig;
18         report ();

19 upon deliver(p, c, [SUMMARY, phashes, msgsig]) ∧ p ∈ missing ∩ P \ prodset_c do
20   if verifysig(p, SUMMARY||phashes, msgsig) then
21     missing := missing \ {p};
22     if correcthashvec = ⊥ then
23       hashvecs[p] := ⟨phashes, msgsig⟩;
24       correcthashvec := minimumHashes(hashvecs);
25       if correcthashvec ≠ ⊥ then
26         consume-and-report ();
27     else if value = ⊥ then
28       if phashes = correcthashvec then
29         hashvecs[p] := ⟨phashes, msgsig⟩;
30         consume-and-report ();
31     else if phashes = correcthashvec then
32       correctproducers[p] := msgsig;
33       report ();
```

of $\mathcal{I} \setminus \{\text{TO}\}$, which contains all the possible coalitions. Each coalition $t \in \mathcal{T}$ may contain simultaneously producers and consumers, represented by $t_{\mathcal{P}} = t \cap \mathcal{P}$ and $t_{\mathcal{C}} = t \cap \mathcal{C}$, respectively. $\Sigma_{\mathcal{I}}$ is a set containing all the profiles of pure strategies $\boldsymbol{\sigma}_{\mathcal{I}}$ followed by all players of $\mathcal{I}$. Σ_t for $t \in \mathcal{T}$ denotes the set of all collusion strategies the players of t may follow. $\succeq_t$ is a preference relation on $\Sigma_{\mathcal{I}} \times \Sigma_{\mathcal{I}}$. We assume that $\succeq_t$ is transitive and reflexive. We can define the relation of strict preference $\succ_t$ as: for any two profiles of strategies $\boldsymbol{\sigma}^*_{\mathcal{I}}, \boldsymbol{\sigma}'_{\mathcal{I}} \in \Sigma_{\mathcal{I}}$, $\boldsymbol{\sigma}^*_{\mathcal{I}} \succ_t \boldsymbol{\sigma}'_{\mathcal{I}}$ iff $\neg(\boldsymbol{\sigma}'_{\mathcal{I}} \succeq_t \boldsymbol{\sigma}^*_{\mathcal{I}})$. If $\boldsymbol{\sigma}^*_{\mathcal{I}} \succ_t \boldsymbol{\sigma}'_{\mathcal{I}}$, then all the players of t will always follow $\boldsymbol{\sigma}^*_{\mathcal{I}}$ over $\boldsymbol{\sigma}'_{\mathcal{I}}$. u_i is the utility function of each player $i \in \mathcal{I}$, defined as $u_i(\boldsymbol{\sigma}_{\mathcal{I}}) = \beta_i(\boldsymbol{\sigma}_{\mathcal{I}}) - \alpha_i(\boldsymbol{\sigma}_{\mathcal{I}})$, where $\beta_i(\boldsymbol{\sigma}_{\mathcal{I}})$ are the benefits and $\alpha_i(\boldsymbol{\sigma}_{\mathcal{I}})$ the costs i incurs when players obey $\boldsymbol{\sigma}_{\mathcal{I}}$.

Sometimes, we will denote the composition of two profiles $\boldsymbol{\sigma}_A$ and $\boldsymbol{\sigma}_B$ as $\boldsymbol{\sigma}_{A \cup B} = (\boldsymbol{\sigma}_A, \boldsymbol{\sigma}_B)$, where A and B are any two disjoint sets of players. Conversely, $u_i(\boldsymbol{\sigma}_A, \boldsymbol{\sigma}_B)$ is equivalent to $u_i(\boldsymbol{\sigma}_{A \cup B})$. Each producer p obtains a benefit $\beta_{\mathcal{P}}$ iff *hasProd*(evidence,p) eventually becomes true, whereas each consumer c obtains a benefit $\beta_{\mathcal{C}}$ iff *hasAck*(evidence,c) eventually becomes true. It is assumed that for all $p \in \mathcal{P}$, $\beta_{\mathcal{P}} > \alpha_p(\boldsymbol{\sigma}_{\mathcal{I}})$, and for all $c \in \mathcal{C}$, $\beta_{\mathcal{C}} > \alpha_c(\boldsymbol{\sigma}_{\mathcal{I}})$, where $\boldsymbol{\sigma}_{\mathcal{I}}$ is the profile of strategies where all players follow the algorithm.

Algorithm 4. NBART (trusted observer TO)

```
01 upon init do
02    evidence := [⊥]^{N_C};

03 upon deliver(c, TO, [REPORT, hashesvec, producers, signature]) do
04    if verifySig(c, REPORT||hashesvec||producers, signature) then
05       evidence[c] := ⟨hashesvec,producers⟩;
06       certify(TO, evidence);
```

A coalition t is said to be Rational if the preference relation $\succeq_t$ fulfils the following condition:

$$\forall_{i \in t} \forall_{\boldsymbol{\sigma}_{\mathcal{I}} \in \Sigma_{\mathcal{I}}, \boldsymbol{\sigma}^*_t \in \Sigma_t} u_i(\boldsymbol{\sigma}_{\mathcal{I}}) \geq u_i(\boldsymbol{\sigma}^*_t, \boldsymbol{\sigma}_{\mathcal{I} \setminus t}) \Rightarrow (\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{\mathcal{I} \setminus t}) \succeq_t (\boldsymbol{\sigma}^*_t, \boldsymbol{\sigma}_{\mathcal{I} \setminus t}).$$

We assume that the same relation holds, by only replacing $\geq$ for $>$ and $\succeq_t$ for $\succ_t$. It follows that if $\#t = 1$ and the only player $i \in t$ is Rational, then for any two profiles of strategies $\boldsymbol{\sigma}^*_{\mathcal{I}}, \boldsymbol{\sigma}'_{\mathcal{I}} \in \Sigma_{\mathcal{I}}$, $\boldsymbol{\sigma}^*_{\mathcal{I}} \succeq_t \boldsymbol{\sigma}'_{\mathcal{I}}$ iff $u_i(\boldsymbol{\sigma}^*_{\mathcal{I}}) \geq u_i(\boldsymbol{\sigma}'_{\mathcal{I}})$. On the contrary, if $\#t = 1$ and the player $i \in t$ is Altruistic, then t is also said to be Altruistic and it is true that $(\boldsymbol{\sigma}_t, \boldsymbol{\sigma}^*_{\mathcal{I} \setminus t}) \succ_t (\boldsymbol{\sigma}^*_{\mathcal{I}})$ for all $\boldsymbol{\sigma}^*_{\mathcal{I}} \in \Sigma_{\mathcal{I}}$ and considering that $\boldsymbol{\sigma}_{\mathcal{I}}$ denotes the profile of strategies where all players follow the algorithm. In any other case, t is Byzantine, implying that $\succeq_t$ is arbitrary due to the Byzantine behaviour of some player from t. It is important to notice that, if t is Byzantine, then all players of t are also considered to be Byzantine, even if some of them have Rational intentions. A coalition t is said to be a producer ($t \in \mathcal{T}_{\mathcal{P}}$) if $t_{\mathcal{P}} \neq \emptyset$ and it is said to be a consumer ($t \in \mathcal{T}_{\mathcal{C}}$) if $t_{\mathcal{C}} \neq \emptyset$.

For simplicity, we model Byzantine behaviour as a single coalition composed by up to $F_{\mathcal{P}} + F_{\mathcal{C}}$ players. We consider an arbitrary number of non-Byzantine coalitions, as long as each coalition is never composed by more than $N^{\mathcal{P}}_{\mathcal{T}}$ producers and $N^{\mathcal{C}}_{\mathcal{T}}$ consumers. As it will be shown later, we now require the following conditions to hold for the algorithm to be tolerant to collusion: $N_{\mathcal{P}} \geq \max(F_{\mathcal{P}}, N^{\mathcal{P}}_{\mathcal{T}}) + F_{\mathcal{P}} + 1$ and $N_{\mathcal{C}} \geq F_{\mathcal{C}} + N^{\mathcal{C}}_{\mathcal{T}} + 1$.

4.2 Expected Utility and Solution Concept

We use the notion of Byzantine-aware utility function for risk-averse players introduced in [10]. An improvement of this work for models where players may be risk-seekers is left for future work. Let $\mathcal{F}_{\mathcal{P}}$ and $\mathcal{F}_{\mathcal{C}}$ denote the set of Byzantine producers and consumers, respectively, and let $\boldsymbol{\pi}_{\mathcal{P}} \in \Pi_{\mathcal{P}}$ and $\boldsymbol{\pi}_{\mathcal{C}} \in \Pi_{\mathcal{C}}$ be the corresponding profiles of strategies. Let us denote by $\boldsymbol{\sigma}_{\mathcal{I} \setminus \mathcal{F}, \boldsymbol{\pi}_{\mathcal{C}}, \boldsymbol{\pi}_{\mathcal{C}}}$ the profile of strategies where all non-Byzantine players follow the strategy specified by $\boldsymbol{\sigma}_{\mathcal{I}}$, Byzantine producers follow the strategies of $\boldsymbol{\pi}_{\mathcal{P}}$ and Byzantine consumers obey the strategies of $\boldsymbol{\pi}_{\mathcal{C}}$. The expected utility of each player $i \in \mathcal{I} \setminus \mathcal{F}$ is defined as follows:

$$\bar{u}_i(\boldsymbol{\sigma}_{\mathcal{M}}) = \min_{\mathcal{F}_{\mathcal{P}}: \#\mathcal{F}_{\mathcal{P}} \leq F_{\mathcal{P}}, \mathcal{F}_{\mathcal{C}}: \#\mathcal{F}_{\mathcal{C}} \leq F_{\mathcal{C}}} \circ \min_{\boldsymbol{\pi}_{\mathcal{P}} \in \Pi_{\mathcal{P}}, \boldsymbol{\pi}_{\mathcal{C}} \in \Pi_{\mathcal{C}}} \circ u_i(\boldsymbol{\sigma}'_{M \setminus \mathcal{F}, \boldsymbol{\pi}_{\mathcal{C}}, \boldsymbol{\pi}_{\mathcal{C}}}). \quad (1)$$

Recall that, since we consider communication costs, a solution concept as strong as (k,t)-robustness is impossible in our case. To overcome this impossibility result, we use the concept of k-resilience combined with the Byzantine aware utility function defined above. However, we still cannot ensure that no player from a coalition t can increase its utility regardless of whether some other player obtains a lower utility or not. What we intend to show is that, regardless of the preferred collusion strategy of each coalition, the chosen strategies fulfil the NBART properties.

In order to formalise this intuition, we define the *observable behaviour* of each coalition $t \in \mathcal{T}$ for the profile of strategies $\boldsymbol{\sigma}_t$ as a multi-set of events triggered in each player $i \in \mathcal{I} \setminus t$ that are influenced by $\boldsymbol{\sigma}_t$, which we denote by $\phi_i(\boldsymbol{\sigma}_t)$. For any player $i \in \mathcal{I} \setminus t$, the delivery of a message sent by some player $j \in t$ is an event. In addition, there are two events triggered in TO, namely *produce*(p,v) for each $p \in t_{\mathcal{P}}$ and *consume*(c,v) for each $c \in t_{\mathcal{C}}$. Henceforth, the meaning of a producer producing a value or a consumer consuming a value is that the corresponding event is eventually triggered in TO.

We say that collusion profile $\boldsymbol{\sigma}_t^* \in \Sigma_t$ is compliant with the profile $\boldsymbol{\sigma}_{\mathcal{I}} = (\boldsymbol{\sigma}_t, \boldsymbol{\sigma}_{\mathcal{I}\setminus t})$ if $\forall_{i \in \mathcal{I}\setminus t} \phi_i(\boldsymbol{\sigma}_t^*) = \phi_i(\boldsymbol{\sigma}_t)$. The set of profiles of strategies compliant with $\boldsymbol{\sigma}_{\mathcal{I}}$ is denoted by $\Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$, where $\boldsymbol{\sigma}_t \in \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$. The solution concept we use in this work, named n *collusion tolerance* (n-cotolerance), is similar to the concept of k-resilience, aside from the fact that we do not require that players in collusion follow the algorithm exactly; only that they follow a profile of strategies from $\Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$. More precisely:

Definition 1. *For any $n \in \mathbb{N}$, a profile of strategies $\boldsymbol{\sigma}_{\mathcal{I}}$ is n-cotolerant iff for all $t \in \mathcal{T}$ such that $\#t \leq n$, for all $\boldsymbol{\sigma}_t^* \in \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$ such that $(\boldsymbol{\sigma}_t^*, \boldsymbol{\sigma}_{\mathcal{I}\setminus t}) \succeq_t \boldsymbol{\sigma}_{\mathcal{I}}$, and for all $\boldsymbol{\sigma}_t' \in \Sigma_t \setminus \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$, $(\boldsymbol{\sigma}_t^*, \boldsymbol{\sigma}_{\mathcal{I}\setminus t}) \succ_t (\boldsymbol{\sigma}_t', \boldsymbol{\sigma}_{\mathcal{I}\setminus t})$.*

The above definition is generic and may be of independent interest. In order to apply it to the NBART problem, we additionally need to capture the distinction between producers and consumers. Therefore, we introduce two parameters $x, y \in \mathbb{N}$, that establish the limit on the number of producers and consumers within the coalition respectively, such that $n \geq x, y$ and $n \leq x + y$. With this definition, if $n = 1$, then there is no collusion among non-Byzantine players. Henceforth, we will say that a profile of strategies $\boldsymbol{\sigma}_{\mathcal{I}}$ is (n,x,y)-cotolerant iff it is n-cotolerant, $n \geq x, y$ and $n \leq x + y$, and for all $t \in \mathcal{T}$ $\#t_{\mathcal{P}} \leq x$ and $\#t_{\mathcal{C}} \leq y$.

4.3 Tolerance to Collusion

The purpose of this section is twofold: i) show that, considering that $\boldsymbol{\sigma}_{\mathcal{I}}$ denotes the profile of strategies where all players follow the algorithm, for any combination of Byzantine and Rational collusions, and any coalition t, if all players of t follow a profile of strategies from $\Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$, then the NBART properties are fulfilled; and ii) show that any profile of strategies $\boldsymbol{\sigma}_t^* \in \Sigma_t$ is preferable to $\boldsymbol{\sigma}_t$ only if $\boldsymbol{\sigma}_t^* \in \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$. The full proofs, included in [16], rely on the assumption that $N_{\mathcal{P}} \geq \max(F_{\mathcal{P}}, N_{\mathcal{T}}^{\mathcal{P}}) + F_{\mathcal{P}} + 1$ and $N_{\mathcal{C}} \geq F_{\mathcal{C}} + N_{\mathcal{T}}^{\mathcal{C}} + 1$.

We show in the following theorem that if each coalition t follows a strategy from $\Sigma_t(\boldsymbol{\sigma}_t^*)$, then the algorithm tolerates collusion. Fix any arbitrary $f \in \mathcal{F}$ such that $\#f_{\mathcal{P}} \leq F_{\mathcal{P}}$ and $\#f_{\mathcal{C}} \leq F_{\mathcal{C}}$. Let l denote an arbitrary partition of $\mathcal{I} \setminus (\{\text{TO}\} \cup f)$ such that, for any $t \in l$, $\#t_{\mathcal{P}} \leq N_{\mathcal{T}}^{\mathcal{P}}$ and $\#t_{\mathcal{C}} \leq N_{\mathcal{T}}^{\mathcal{C}}$. Furthermore, let $\boldsymbol{\sigma}_{\mathcal{I}}^* = ((\boldsymbol{\sigma}_t^*)_{t \in l, \boldsymbol{\sigma}_t^* \in \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})}, (\pi_p)_{p \in f_{\mathcal{P}}}, (\pi_c)_{c \in f_{\mathcal{C}}})$.

Theorem 1. *For some arbitrary partition l and profile $\boldsymbol{\sigma}_{\mathcal{I}}^*$, if all players follow $\boldsymbol{\sigma}_{\mathcal{I}}^*$, then the NBART properties are fulfilled.*

The following two lemmas show that, for each $t \in \mathcal{T}$ the expected benefit is 0 for all $i \in t$, whenever players of t follow a profile of strategies from $\Sigma_t \setminus \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$. Recall that we assume that the players are risk averse. Therefore, the analysis is done assuming worst case Byzantine behaviour.

Lemma 1. *For any $t \in \mathcal{T}_{\mathcal{C}}$, let $\boldsymbol{\sigma}'_t \in \Sigma_t \setminus \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$ be any profile of strategies where t does not ensure that for all $c \in t_{\mathcal{C}}$ and $p \in \mathcal{P}$, c certifies p and invokes* consume(c, v)*, and, for each, $p \in t_{\mathcal{P}}$ p invokes* produce(p, v)*. Then, for all $i \in t$ $\bar{\beta}_i(\boldsymbol{\sigma}'_t, \boldsymbol{\sigma}_{\mathcal{I} \setminus t}) = 0$.*

Lemma 2. *For all $t \in \mathcal{T}$, $i \in t$, and $\boldsymbol{\sigma}'_t \in \Sigma_t \setminus \Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}})$, $\bar{\beta}_i(\boldsymbol{\sigma}'_t, \boldsymbol{\sigma}_{\mathcal{I} \setminus t}) = 0$.*

The following theorem concludes that the proposed algorithm is $(N_{\mathcal{T}}^{\mathcal{P}} + N_{\mathcal{T}}^{\mathcal{C}}, N_{\mathcal{T}}^{\mathcal{P}}, N_{\mathcal{T}}^{\mathcal{C}})$-cotolerant.

Theorem 2. *Let $\boldsymbol{\sigma}_{\mathcal{I}} \in \Sigma_{\mathcal{I}}$ denote the profile of strategies where all players follow the algorithm. Then, $\boldsymbol{\sigma}_{\mathcal{I}}$ is $(N_{\mathcal{T}}^{\mathcal{P}} + N_{\mathcal{T}}^{\mathcal{C}}, N_{\mathcal{T}}^{\mathcal{P}}, N_{\mathcal{T}}^{\mathcal{C}})$-cotolerant.*

A particular case of this result is that $\boldsymbol{\sigma}_{\mathcal{I}}$ is $(1, 1, 1)$-cotolerant. By the definition of $\succeq_t$ for any $t \in \mathcal{T}$ such that $\#t = 1$, $\Sigma_t(\boldsymbol{\sigma}_{\mathcal{I}}) = \{\boldsymbol{\sigma}_{\mathcal{I}}\}$, implying that $\boldsymbol{\sigma}_{\mathcal{I}}$ is a *Nash equilibrium.*

Acknowledgements. This work was partially supported by the FCT (INESC-ID multi annual funding through the PIDDAC Program fund grant and by the project PTDC/EIA-EIA/102212/2008).

References

1. Anderson, D.: Boinc: A system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID 2004, Pittsburgh, PA, USA, pp. 4–10. IEEE (November 2004)
2. Aiyer, S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.P., Porth, C.: BAR fault tolerance for cooperative services. In: Proceedings of the 20th ACM Symposium on Operating Systems Principles, SOSP 2005, Brighton, United Kingdom, pp. 45–58. ACM (October 2005)
3. Vilaça, X., Leitão, J., Correia, M., Rodrigues, L.: N-party BAR Transfer. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 392–408. Springer, Heidelberg (2011)
4. Eliaz, K.: Fault-tolerant implementation. Review of Economic Studies 69(3), 589–610 (2002)

5. Moscibroda, T., Schmid, S., Wattenhofer, R.: On the topologies formed by selfish peers. In: Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, pp. 133–142. ACM (July 2006)
6. Aumann, R.J.: Acceptable points in General Cooperative n-person Games. In: Contributions to the Theory of Games IV. Annals of Mathematics Studies, vol. (40), pp. 287–324. Princeton University Press, Princeton (1959)
7. Bernheim, B., Peleg, B., Whinston, M.: Coalition-proof nash equilibria i. concepts. Journal of Economic Theory 42(1), 1–12 (1987)
8. Moreno, D., Wooders, J.: Coalition-proof equilibrium. Games and Economic Behavior 17(1), 80–112 (1996)
9. Abraham, I., Dolev, D., Gonen, R., Halpern, J.: Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In: Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, pp. 53–62. ACM (July 2006)
10. Clement, A., Napper, J., Li, H., Martin, J.P., Alvisi, L., Dahlin, M.: Theory of BAR games. In: Proceedings of the 26th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2007, Portland, OR, USA, pp. 358–359. ACM (August 2007)
11. Li, H., Clement, A., Wong, E., Napper, J., Roy, I., Alvisi, L., Dahlin, M.: BAR gossip. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2006, Seattle, WA, USA, pp. 191–204. USENIX Association (November 2006)
12. Li, H., Clement, A., Marchetti, M., Kapritsos, M., Robison, L., Alvisi, L., Dahlin, M.: Flightpath: Obedience vs choice in cooperative services. In: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, San Diego, CA, USA, pp. 355–368. USENIX Association (December 2008)
13. Mokhtar, S., Pace, A., Quéma, V.: FireSpam: Spam resilient gossiping in the BAR model. In: Proceedings of the 29th IEEE International Symposium on Reliable Distributed Systems, SRDS 2010, New Delhi, India, pp. 225–234. IEEE (October 2010)
14. Wong, E.L., Leners, J.B., Alvisi, L.: It's on Me! The Benefit of Altruism in BAR Environments. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 406–420. Springer, Heidelberg (2010)
15. Cachin, C., Guerraoui, R., Rodrigues, L.: Introduction to Reliable and Secure Distributed Programming, 2nd edn. Springer-Verlag New York, Inc. (2011)
16. Vilaça, X., Denysyuk, O., Rodrigues, L.: Asynchrony and collusion in the N-party BAR transfer problem. Arxiv preprint arXiv:1204.4044v1 (2012)

Early Deciding Synchronous Renaming in $O(\log f)$ Rounds or Less

Dan Alistarh[1], Hagit Attiya[2,⋆], Rachid Guerraoui[3], and Corentin Travers[4,⋆⋆]

[1] EPFL
alistarh@epfl.ch
[2] Technion
hagit@cs.technion.ac.il
[3] EPFL
guerraoui@epfl.ch
[4] Univ. Bordeaux
travers@labri.fr

Abstract. Renaming is a fundamental problem in distributed computing, in which a set of n processes need to pick unique names from a namespace of limited size. In this paper, we present the first *early-deciding* upper bounds for synchronous renaming, in which the running time adapts to the actual number of failures f in the execution. We show that, surprisingly, renaming can be solved in *constant* time if the number of failures f is limited to $O(\sqrt{n})$, while for general $f \leq n-1$ renaming can always be solved in $O(\log f)$ communication rounds. In the wait-free case, i.e. for $f = n-1$, our upper bounds match the $\Omega(\log n)$ lower bound of Chaudhuri et al. [13].

1 Introduction

Unique names, or identifiers, are a fundamental prerequisite for solving a variety of problems in distributed computing. While in many settings unique names are available, they often come from a very large, practically unbounded namespace, which reduces their usefulness. The *renaming* problem [4], in which a set of processes need to be assigned names from a namespace of small size, is one of the fundamental problems in distributed computing, and a significant amount of work, e.g. [1,2,4,6,7,8,9,10,17,18], studied its solvability and complexity in fault-prone distributed systems.

Much of the work on the renaming problem, whether in shared-memory [1,6] or in message-passing systems [4], has focused on the asynchronous case, in which the processes' steps or messages may be delayed arbitrarily by the scheduler. However, real world systems experience long periods where delays are bounded, and inter-process communication is synchronous, even though processes may still fail by crashing. The complexity of renaming in such a synchronous setting, where processes communicate by message-passing, was investigated by Chaudhuri et al. [13] and Okun [19]. In brief, they found that for n processes,

⋆ Supported in part by the *Israel Science Foundation* (grant number 1227/10).
⋆⋆ Additional supports from the ANR projects ALADDIN and DISPLEXITY.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 195–206, 2012.

up to $n-1$ of which may fail by crashing, there exist algorithms that assign a tight namespace of names $1,\ldots,n$ names in $O(\log n)$ rounds of communication. Chaudhuri et al. [13] also showed a matching lower bound.

The analysis in these papers only focused on the case where the *maximum* number of processes, $n-1$, may fail by crashing during an execution. However, the $\Omega(\log n)$ lower bound of [13] does not exclude algorithms that terminate faster when the *actual* number of failures f in the execution is less than $n-1$. Such speculative algorithms, also known as *early deciding*, are known to exist for *consensus* [20] and *set agreement* [12]. Early-deciding protocols achieve consensus in $O(f)$ communication rounds [15], and k-set agreement in $O(\frac{f}{k})$ rounds [16].

It is therefore natural to ask what is the time complexity of early deciding synchronous renaming in a message-passing system. The answer to this question does not appear trivial: for example, a successful strategy for renaming [13] was for each process to obtain a new bit from its name by running a simple one round symmetry-breaking protocol–after $\Theta(\log n)$ rounds, each process has a unique name from 1 to n. However, it is hard to speed up this approach to obtain more bits for the process's name in rounds where there are few failures, without breaking name uniqueness. Another approach [4], where each process proposes a name in each round based on what every other process proposed in previous rounds, until there are no collisions, turns out to be very difficult to analyze when the adversary has a limited budget of f failures.

In this paper, we overcome these challenges, and present the first early-deciding upper bounds for renaming. In short, we find that the complexity of the problem is strongly coupled with the relation between n, the number of processes and f, the failure budget of the adversary. We show that there exists an algorithm that ensures a *tight* namespace of n names, and terminates in a *constant* number of rounds in every execution where $f \in O(\sqrt{n})$, and, in $O(\log f)$ rounds, otherwise. The existence of a constant-time renaming algorithm for non-trivial f is surprising, since the early-deciding bounds for consensus [15] and set agreement [16] are *linear* in f irrespective of the relation with the total number of processes n.

Our second result is an algorithm that improves on the constants in the asymptotic notation, terminating in $\log f + 5$ rounds, and assigning names from 1 to $2n$.

The first algorithm is a slight modification of a result by Okun [19]. His protocol is based on a novel connection between synchronous renaming and the approximate agreement problem [14]. In brief, processes assign temporary ranks to each process identifier that they receive, and perform approximate agreement to converge on an approximate rank for each initial process identifier, within a carefully chosen approximation factor. Each process returns this approximate rank, rounded to the nearest integer, as its name: the protocol ensures that, upon termination, the approximate ranks are far enough apart so that no two processes decide on the same rank. Okun [19] showed that this protocol ensures a tight namespace, preserves the order of the initial identifiers, and terminates in $O(\log n)$ rounds in all executions.

Our main contribution is analyzing this protocol for general f, and showing that it terminates in *constant* time if $f \in O(\sqrt{n})$, and in $O(\log f)$ rounds otherwise. Our analysis characterizes the optimal adversarial strategy for arbitrary values of the parameters n and f—we achieve this by carefully bounding the approximation factor of the approximate agreement protocols relative to the number of failures that the adversary expends in each round, showing that this factor goes down fast, and the algorithm terminates very quickly, if the adversary does not fail a significant fraction of the processes in each round.

The second renaming algorithm we present is simpler, and achieves better constants in the asymptotic notation, while relaxing the size of the namespace to $2n$. The protocol is split in two phases: In the first phase, each process identifies a set of names it is interested in, whose size is proportional to the number of failures that the adversary expends in the phase. In the second phase, processes proceed to progressively halve the size of the set of names they are interested in, until each set is a singleton. At this point, each process may adopt the single name in its set. The key technical difficulty is in assigning each process the "right" set of names at the end of the first phase, so that there are always enough names for the set of participants. To ensure this, we need to relax the total size of the namespace that processes are interested in to be of size $2n$. The halving procedure in the second phase is similar to the $O(\log n)$ round algorithm by Chaudhuri et al. [13].

We therefore show that the time complexity of early-deciding renaming is upper bounded by $O(\log f)$ synchronous rounds for general f, and can be constant for non-trivial $f \in O(\sqrt{n})$. Both algorithms are *adaptive*, since they do not know the number of participants n in advance; they also adapt to the number of failures f in the execution.

Roadmap. We present the problem statement and the model in Section 2, and give an overview of related work in Section 3. In Section 4, we present the analysis of the tight renaming algorithm, while the second algorithm can be found in Section 5. We conclude with a discussion of open questions in Section 6.

2 Model and Problem Statement

Model. We consider a standard synchronous message passing system with n processes $p_1, \ldots, p_n$. Initially, processes have unique identifiers from a namespace of unbounded size. Time is divided into *rounds*, and the processes' clocks are synchronized. Each round proceeds as a sequence of send, receive, and process steps, in which processes may send and receive messages, and perform local computation if necessary. We assume that at most $t < n$ processes may fail by crashing. If crashed, a process stops taking further steps in the execution; a process may fail to send any subset of its messages in rounds in which it crashes.

Let f denote the number of failures in the current execution. We focus on *early-deciding* algorithms, that adapt their time complexity to the actual number of failures in the execution, i.e. whose running time is a function of f only.

Renaming. The *renaming* problem [4] requires that each correct process eventually returns a name, and that the names returned should be unique. The size of the resulting namespace should only depend on the parameters n. The *tight* renaming problem requires that the size of the namespace be exactly n; otherwise, we say that the solution is *loose*.

Approximate Agreement. Consider a small real number $\epsilon > 0$. In the ϵ-*approximate agreement* problem [14], each process p_i starts with a proposal v_i, which is a real number. An approximate agreement algorithm must satisfy the following properties: (1) Each correct process eventually decides a value d_i; (2) For any correct processes p_i and p_j, $|d_i - d_j| \leq \epsilon$; (3) For any correct process p_ℓ, there are processes p_i and p_j with initial values v_i and v_j, such that $v_i \leq d_\ell \leq v_j$.

Notation. Throughout this paper, log denotes the logarithm base two. Moreover, we write $\log f$ instead of $\lfloor \log f \rfloor$. To simplify notation, we assume that $f \geq 1$, which allows us to consider running times of $O(\log f)$ rounds.

3 Related Work

The renaming problem was introduced in [4], where the authors also provide a wait-free solution using $(2n-1)$ names in an asynchronous message-passing system, and show that at least $(n+1)$ names are required in the wait-free case. This lower bound on the namespace size for the case of asynchronous solutions was improved to $(2n-2)$ by Herlihy and Shavit [17], and Rajsbaum and Castañeda [11]. (This lower bound holds when n is a power of a prime number.)

Considerable research has analyzed the upper and lower bounds for renaming in asynchronous setting, in particular in shared memory e.g., [1,2,6,7,8,9,18]. For a detailed description of these results, we refer the reader to e.g., [1].

On the other hand, there has been relatively little work on the complexity of synchronous renaming. Herlihy et al. [13] considered wait-free renaming in a synchronous message-passing system, identical to the one we consider in this paper. They prove a lower bound of $\Omega(\log n)$ rounds on the time complexity of the problem in runs where $t = n-1$ processes may crash, and provide a matching algorithm that achieves tight renaming with time complexity $\lfloor \log n \rfloor + 3$ in *every* execution. In contrast, our algorithms are *early deciding*, in that they adapt to the *actual* number of failures f in the current execution, deciding in $O(\log f)$ rounds. On the other hand, the lower bound argument of Herlihy et al. [13] applies to our algorithms as well[1], implying a lower bound of $\Omega(\log n)$ rounds in executions where $f = n-1$.

In [19], Okun presents a synchronous message-passing algorithm for tight renaming algorithm with $O(\log n)$ time complexity. His algorithm is based on a

[1] While the original lower bound of Herlihy et al. [13] applies only to comparison-based algorithms, an argument by Attiya et al. [5] generalizes this bound to a wider class of algorithms, which includes the ones in this paper.

```
1 procedure propose_i(v_i, ε) ;
2   d_i ← v_i;
3   for each round r ≥ 0 do
4       broadcast(d_i);
5       δ ← max_{ℓ≠j}(|d_ℓ − d_j|);
6       if δ ≤ ε then return d_i;
7       else d_i ← arithmetic average of all values d_j received;
```

Fig. 1. The Approximate Agreement algorithm

novel connection between renaming and approximate agreement. In brief, processes perform approximate agreement on the rank of each initial identifier, until they are certain that no two processes obtain the same rank. In this paper, we extend this technique by providing a new analysis for minor variation of his algorithm, proving that its running time is in fact $O(\log f)$ in executions with f failures, which is asymptotically optimal.

4 A Tight Renaming Algorithm

In this section, we analyze the tight renaming algorithm of Okun [19] and prove that it terminates in $O(\log f)$ rounds, where $f < n$ is the number of processes that the adversary crashes in the current execution. We begin with a short description of the algorithm; a detailed exposition can be found in the original paper [19].

Algorithm Overview. The algorithm is based on a novel connection between renaming and the approximate agreement problem. First, a simple synchronous approximate agreement (AA) algorithm is introduced. Then, the algorithm runs in parallel at most n separate instances of this approximate agreement algorithm, one for *each* process in the system. The goal is to agree on an approximate rank for each process's initial identifier, which will be the value decided by the corresponding approximate agreement algorithm, rounded to the nearest integer. The key ingredient is to run the approximate agreement algorithm for long enough to ensure that the decision values of the AA protocols corresponding to each initial identifier are sufficiently spaced, ensuring that the rank decided for each process is unique.

The Approximate Agreement Algorithm. The algorithm, whose pseudocode appears in Figure 1, proceeds as follows. Each process starts with an initial value v_i and a desired approximation factor ϵ, which bounds the maximum desired skew between decided values. The process maintains an estimate d_i of its decision value, which is updated in every round to the arithmetic average of all values received. Once all the estimates received in a round are within at most ϵ of each other, the process returns its current estimate.

```
1  procedure rename_i(v_i) ;
     /* Phase one */
2    for each round r = 1, 2, 3 do
3      broadcast(v_i);
4      C_r ← the number of distinct identifiers received in round r;
5    n ← C_1;
6    V ← the set of identifiers received in round 3;
7    ε ← 1/(10C_2);
     /* Phase two */
8    for each round r ≥ 4 do
9      for every identifier id ∈ V do
10       Participate in id's instance of the approximate agreement algorithm,
11       with initial value C_2 · rank_V(id) , until the algorithm returns;
         /* rank_V(id) is the rank of id in the set V, in increasing order */
     /* Upon completion of all the approximate agreement algorithms */
12   name_i ← final value in v_i's instance of the approximate agreement algorithm,
     rounded to the nearest integer;
13   return name_i;
```

Fig. 2. The Tight Renaming Algorithm

The Renaming Algorithm. Each process p_i starts with an initial name v_i. The algorithm, whose pseudocode appears in Figure 2, has two phases.

The first phase contains the first three rounds, in which processes exchange their identifiers, in order to identify the parameter n and the relative ranks of their identifiers.

In the second phase, which starts at round four, based on the information computed so far, each process proposes a rank for each participating process to a separate instance of the approximate agreement algorithm in Figure 1. Notice that all these agreement instances run in parallel; all messages by a process in a round (one for each AA protocol in which it participates) are packaged into a single composite message, which each process sends in each round. The approximation factor for all these agreement instances is $1/(10C_2)$, where C_2 is the number of distinct identifiers the process received in round 2 of the first phase. (This factor is chosen such that no two identifiers may receive the same final rank from the approximate agreement instances when rounded to the nearest integer.) The algorithm terminates when all the approximate agreement algorithms terminate, ensuring the desired approximation factor.

Name Uniqueness. We give a brief overview of the mechanism ensuring name uniqueness; a complete analysis can be found in [19]. Recall that, for each initial identifier id, each process p proposes the rank of id multiplied by C_2/C_3 as the initial value in id's instance of the AA protocol. Obviously, the processes' ranks for the same identifier may be distinct (as a consequence of failures in the first phase). However, a key observation is that they will be distinct in a consistent

way: if p proposes rank 6 for id α, and rank 7 for id $\beta > \alpha$, then another process q proposing rank 7 for id α will have to propose rank at least 8 for id β. Analyzing the AA protocol, this will imply that, given any process p, its decision values for distinct ids $\alpha < \beta$ will be at distance at least 1 from each other [19, Theorem 3].

This mechanism might still allow the possibility that two distinct processes decide on the same name when rounding their AA decision value to the nearest integer. This is handled by multiplying the initial ranks with C_2/C_3. This ratio is higher than 1 only when a processor observes crashes between the second and third rounds, and the algorithm ensures that for any ids α and β, their corresponding decision values at any two processes p and q are at distance > 1 from each other [19, Lemma 3]. In turn, this implies that no two processes may decide the same name.

Analysis. Our key observation is that the variant of the synchronous approximate agreement algorithm of Okun [19] presented in Figure 1 guarantees the required approximation factor of $1/(10C_2)$ in a constant number of rounds when $f > \sqrt{n}/2$, and $O(\log f)$ rounds, otherwise. In turn, this implies that the renaming algorithm terminates within three additional rounds.

To prove this, we first introduce some notation. Let $\sigma(S)$ be the *diameter* of a set S, i.e. $\max_{a,b \in S}(|a - b|)$.

For any round $r > 0$ in the execution of the approximate agreement algorithm, let U_r be the multiset of distinct values that processes that are active (i.e., send at least one message) in round r have in the beginning of the round.

Let f_r be the number of processes that crash in round r; for convenience, denote $f_0 = 0$. Denote by δ_r the fraction of processes that crash in round r from among the processes that did not crash before round r; that is, $\delta_r = f_r/(n - \sum_{i=0}^{r-1} f_i)$.

We first state the following lemma, bounding the diameter of the set U_{r+1} depending on the diameter of U_r and the fraction of processes that crash in round r; its proof follows [19, Lemma 1].

Lemma 1. $\sigma(U_{r+1}) \leq \frac{2\delta_{r+1}}{1-\delta_{r+1}} \cdot \sigma(U_r)$.

The next lemma bounds the number of rounds needed for the approximate agreement algorithm of Figure 1 to achieve a maximum diameter of $\epsilon = 1/(10n)$ for the set of decisions corresponding to each initial value, when starting with proposal sets of diameter $\leq n$. The bound depends on f, the number of failures that the adversary expends in total, and on the number of failures f_i which the adversary expends in each round $i \geq 1$.

Lemma 2. *Consider an execution of the approximate agreement algorithm of Figure 1, starting with an initial set of diameter $\sigma(U_1) \leq n$, in which at most f processes crash. Let R be the number of rounds needed for the algorithm to reach a diameter $\epsilon \leq 1/(10n)$. The following claims hold:*

- *If $f \leq \sqrt{n}/2$, then R is a constant.*
- *If $\sqrt{n}/2 < f \leq n - 1$, then $R \leq 5 \log f + 10$.*

Proof. We assume $n \geq 6$; the claim can be checked for $n \leq 5$ by calculation.

The first case is when $f \leq \sqrt{n}/2$. In this case, notice that the fraction of processes that fail in any round of the protocol is at most $\sqrt{n}/n$, i.e. $\delta_r \leq \sqrt{n}/n$, for all $r \geq 1$. In turn, by Lemma 1, the diameter of the set of values for the current instance of approximate agreement is reduced by at least $2\delta_r/(1-\delta_r) \leq 1/(\sqrt{n}-0.5)$ in each round r. Therefore, after the first 10 rounds, the diameter of this set is at most

$$\frac{n}{(\sqrt{n}-0.5)^{10}} \leq \frac{1}{10n}, \text{ for any } n \geq 6.$$

Therefore the maximum diameter the end of round 10 is $\leq 1/(10n)$, as claimed.

In the second case, we assume that $f > \sqrt{n}/2$. First, notice that δ_r, the fraction of active processes that crash in a round r, can be greater than $1/2$ in at most $\lfloor \log_2(f+1) \rfloor + 1$ distinct rounds. Therefore, any execution of at least $5\lfloor \log_2(f+1) \rfloor + 10$ rounds contains at least $4\lfloor \log_2(f+1) \rfloor + 9$ rounds r in which $\delta_r < 1/2$. Lemma 1 implies that the diameter of the set U_r at the end of these rounds is at most

$$\frac{n}{2^{4\lfloor \log_2(f+1) \rfloor + 9}} \leq \frac{n}{2^9 \cdot (f+1)^4} \leq \frac{1}{10n} \text{ for all } n \geq 1,$$

where the last step uses the fact that $f \geq \sqrt{n}/2$. Therefore, in this case, the number of rounds necessary to obtain a maximum diameter of at most $1/(10n)$ is $O(\log f)$. □

We conclude that the resulting renaming algorithm is early deciding, terminating in a constant number of rounds, when $f \leq \sqrt{n}/2$, and $O(\log f)$ rounds, otherwise.

Theorem 1. *For any f, $0 \leq f \leq n-1$, let ℓ_f = constant if $f \leq \sqrt{n}/2$, and $\ell_f = 5\log(f+1) + 10$, otherwise. Then the renaming algorithm in Figure 2 is a tight order-preserving renaming algorithm with time complexity $O(\ell_f)$ and message complexity $O(n^2\ell_f)$ in executions where f processes fail by crashing.*

Proof. We focus on early decision, since the other properties follow from Theorem 4 of [19]. First, recall that the initial value v for each approximate agreement algorithm is computed locally by each process as $\mathsf{rank}_V(\alpha) \cdot C_2/C_3$. Since $\mathsf{rank}_V(\alpha)/C_3 \leq 1$, by the definition of C_2, we obtain that all initial values are between 1 and n. This also implies that our desired approximation factor ϵ is at most $1/(10n)$.

Lemma 2 implies that all instances of approximate agreement that the renaming algorithm executes in parallel terminate in $O(\ell_f)$ rounds, with an approximation factor $\epsilon \leq 1/(10n)$. We obtain that the renaming algorithm terminates in $O(\ell_f)$ rounds, as claimed. □

5 A Loose Renaming Algorithm with Improved Round Complexity

In this section, we present a loose renaming algorithm that terminates in $\log f_1+5$ rounds and uses at most $2n$ names, where n is the number of participating processes and f_1 the number of failures that occur in the first round. Our algorithm extends the non-early deciding renaming algorithm by Chaudhuri, Herlihy, and Tuttle [13]. The latter algorithm is tight; its round complexity, however, depends solely on n and not on the number of failures among the participating processes. Specifically, the namespace when n processes participate is $[1, n]$ and the algorithm terminates in $\log n + 2$ rounds. In the following, the algorithm of [13] is called the *CHT-renaming* algorithm.

Algorithm Overview. The pseudo-code of the algorithm appears in Figure 3. It contains two phases.

In the first phase, which consists of the two first rounds (line 1-line 7), each process selects an interval of names in which it wishes to pick its final name. The size of each interval is upper-bounded by $4f_1$, where f_1 is the number of processes that fail during the first round. The second phase (line 8-line 16) consists of a variant of the CHT-renaming algorithm. Processes use this procedure to progressively shrink the interval of names they are interested in, until obtaining an interval of size 1. The algorithm ensures that no two processes obtain the same interval of size 1, i.e. a single name. Thus, each process can decide the unique name in its final interval. Moreover, it guarantees that in each round, processes holding the largest intervals reduce their interval by one half. Therefore our algorithm terminates in $O(\log f_1)$ rounds.

We begin with a brief description of the CHT-renaming algorithm and then explain how each process selects an initial interval of names.

Definitions and Notations. Before describing the algorithm in more details, we introduce some notations, extending those in [13]. An interval I of positive integers is *well-formed* if I is of the form $[d2^j + 1, (d + 1)2^j]$ for some positive integers d, j. Note that for every pair I, J of well-formed intervals, $I \cap J \neq \emptyset \implies I \subseteq J \vee J \subseteq I$. Given a set $\mathcal{I}$ of well-formed intervals, we say that interval $I \in \mathcal{I}$ is *maximal* in $\mathcal{I}$ if for every $J \in \mathcal{I}$, either $I \cap J = \emptyset$ or $J \subseteq I$.

The CHT-Renaming Algorithm. Each process p maintains a well-formed interval of names I, which are the names p is interested in. In each round, process p sends its id and the interval it currently holds I. The intervals intersecting with I that p receives are stored in the set $\mathcal{I}$ (line 10). p also stores the set of ids of the processes that are conflicting with p, that is, processes that hold an interval intersecting with I in the set $\mathcal{P}$ (line 11). If I is maximal in $\mathcal{I}$ (line 12), I is split in half, and p picks the bottom half or the top half of I, denoted $bot(I)$ and $top(I)$ respectively, as its new interval (line 13-line 14). More precisely, the new interval of p is $bot(I)$ if the rank of p's identity in $\mathcal{P}$ is smaller than $\frac{|I|}{2}$. Otherwise, p selects $top(I)$ as its new interval. Finally, if p observes that every

```
1  procedure rename_i(id_i) ;
       /* Round 1 */
2      broadcast(id_i);
3      rk ← the rank of id among the id received in round 1;
       /* Round 2 */
4      broadcast(id_i, rk);
5      let rk_max = largest rank received; for each j, 1 ≤ j ≤ rk_max do
       h_j ← |{⟨id', rk'⟩ : rk' = j}| ;
6      est_f ← max{(∑_{j∈I} h_j) − |I| : I ⊆ [1, rk_max]};
7      let j = ⌈log(est_f)⌉ and d such that d · 2^j + 1 ≤ rk ≤ (d + 1) · 2^j ;
8      I ← [d · 2^{j+1} + 1, (d + 1) · 2^{j+1}] ;
       /* Round 3, 4, ...: CHT-renaming protocol [13] */
9      repeat
10         broadcast(id, I) ;
11         I ← {I' : ⟨id', I'⟩ received and I ∩ I' ≠ ∅} ;
12         P ← {id' : ⟨id', I'⟩ received and I ∩ I' ≠ ∅} ;
13         if ∀I' ∈ I, I' ⊆ I then
14             let bot(I) and top(I) the bottom half and top half of I respectively;
15             if rank(id, P) ≤ |I|/2 then I ← bot(I) else I ← top(I)
16     until ∀I' ∈ I, |I'| = 1 ;
17     return name_i where I = [name_i, name_i] ;
```

Fig. 3. The Loose Renaming Protocol

interval sent in the round has size 1, it decides the unique name in its interval. Since in each round the size of maximal interval is at least divided by 2, the algorithm terminates after $O(\log c)$ rounds, where c is the size of the largest initial maximal interval.

Name uniqueness relies on the following invariant, which is satisfied by the set of intervals $\mathcal{I}[r]$ held by the processes at the beginning of each round r:

Invariant 1. *For every $I \in \mathcal{I}[r]$, if m processes hold an interval $I' \subseteq I$, then $m \leq |I|$. In particular, this means that I is large enough to allow processes with an interval $I' \subseteq I$ to decide distinct names in the interval I.*

In the original CHT-renaming, the initial interval of each process p is of the form $[1, 2^b]$, where 2^b is the least power of 2 larger than or equal to the number of participating processes from which p has received a message in the first round. The invariant above is thus initially true. In our algorithm, initial intervals are selected differently, but the invariant is still satisfied.

Selection of Initial Intervals. In the first round, process p broadcasts its id, and ranks its id among the ids it receives (line 2). Let rk_p be the rank obtained by p. If, among the ids of the participating processes, the rank of the id of p is i, and f_1 is the number of failures that occur in the first round, then

$$i - f_1 \leq rk_p \leq i. \tag{1}$$

This is because p may miss at most f_1 messages from processes with id smaller than i. In the second round, p sends its rank together with its id. It then estimates, based on the ranks it receives, the number of failures that occur in the first round. To that end, p evaluates for each interval of names I the difference between the number of processes ranked in I and the size of I (line 4-line 5). The estimate est_f of the number of failures is then the maximum over all differences. More precisely,

$$est\_f = \max_{I \subseteq [1, rk\_max]} \sum_{i \in I} h_i - |I|.$$

where rk_max is the largest rank received by p and h_i is—to the knowledge of p—the number of processes that rank their id i in the first round. The estimation est_f is upper-bounded by f_1, the number f_1 of failures that occur in the first round. To see why, consider an interval $I = [a, b]$. We have:

$$\sum_{i \in I} h_i \leq |\{q : a \leq rk_q \leq b\}| \leq |\{q : a - f_1 \leq rank(id_q, P) \leq b\}| \leq |I| + f_1, \quad (2)$$

where P is the set of participating processes and $rank(id_q, P)$ is the rank of id_q among the ids of the processes in P. The last inequality follows from Equation 1.

Finally, p selects a well-formed interval. This is performed in two steps. First, p chooses two integers d, j such that the well-formed interval $[d2^j + 1, (d + 1)2^j]$ contains rk_p, and 2^j is the least power of 2 larger than or equal to est_f. However, by equation (2), at most $2^j + est\_f \leq 2^{j+1}$ processes may have their rank contained in $[d2^j+1, (d+1)2^j]$. Then, in order to satisfy Invariant 1, p selects the interval $I = [d2^{j+1} + 1, (d + 1)2^{j+1}]$ as its initial interval. Since a process p' that selects an interval $\subseteq I$ has its rank $rk_{p'}$ contained in $[d2^j + 1, (d + 1)2^j]$, at most $2^{j+1} = |I|$ processes select intervals $I' \subseteq I$. Invariant 1 is thus satisfied, which preserves the correctness of the CHT-renaming algorithm. Finally, notice that the size of each initial interval is at most $4f_1$, where f_1 is the number of failures in the first round. Hence the total running time is $O(\log f_1)$ rounds.

The proof of correctness can be found in a companion technical report [3].

6 Discussion

This paper presents the first early-deciding upper bounds for synchronous renaming. We show that, surprisingly, renaming can be solved in *constant* number of rounds if the number of failures f is limited to $O(\sqrt{n})$, while in the general case, renaming can always be solved in $O(\log f)$ communication rounds. In the wait-free case, i.e. for $f = n - 1$, this upper bound is matched asymptotically by the $\Omega(\log n)$ lower bound of Chaudhuri et al. [13]. It remains an open question whether this is tight for other values of f.

References

1. Alistarh, D., Aspnes, J., Censor-Hillel, K., Gilbert, S., Zadimoghaddam, M.: Optimal-time adaptive strong renaming, with applications to counting. In: PODC 2011: Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, pp. 239–248 (2011)

2. Alistarh, D., Aspnes, J., Gilbert, S., Guerraoui, R.: The complexity of renaming. In: FOCS 2011: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, pp. 718–727 (2011)
3. Alistarh, D., Attiya, H., Guerraoui, R., Travers, C.: Early deciding synchronous renaming in $O(\log f)$ rounds or less. Technical report, INRIA (2012), http://hal.inria.fr/hal-00687555/en/
4. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. Journal of the ACM 37(3), 524–548 (1990)
5. Attiya, H., Djerassi-Shintel, T.: Time bounds for decision problems in the presence of timing uncertainty and failures. Journal of Parallel and Distributed Computing 61(8), 1096–1109 (2001)
6. Attiya, H., Fouren, A.: Adaptive and efficient algorithms for lattice agreement and renaming. SIAM Journal on Computing 31(2), 642–664 (2001)
7. Borowsky, E., Gafni, E.: Immediate atomic snapshots and fast renaming. In: PODC 2093: Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing, pp. 41–51. ACM, New York (1993)
8. Brodsky, A., Ellen, F., Woelfel, P.: Fully-adaptive algorithms for long-lived renaming. Distributed Computing 24(2), 119–134 (2011)
9. Burns, J.E., Peterson, G.L.: The ambiguity of choosing. In: PODC 1989: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, pp. 145–157. ACM, New York (1989)
10. Castañeda, A., Rajsbaum, S.: New combinatorial topology bounds for renaming: The upper bound. Journal of the ACM 59(1) (March 2012)
11. Castañeda, A., Rajsbaum, S.: New combinatorial topology bounds for renaming: The lower bound. Distributed Computing 22(5-6), 287–301 (2010)
12. Chaudhuri, S.: More choices allow more faults: Set consensus problems in totally asynchronous systems. Information and Computation 105(1), 132–158 (1993)
13. Chaudhuri, S., Herlihy, M., Tuttle, M.R.: Wait-free implementations in message-passing systems. Theoretical Computer Science 220(1), 211–245 (1999)
14. Dolev, D., Lynch, N.A., Pinter, S.S., Stark, E.W., Weihl, W.E.: Reaching approximate agreement in the presence of faults. Journal of the ACM 33, 499–516 (1986)
15. Dolev, D., Reischuk, R., Raymond Strong, H.: Early stopping in byzantine agreement. Journal of the ACM 37(4), 720–741 (1990)
16. Gafni, E., Guerraoui, R., Pochon, B.: The complexity of early deciding set agreement. SIAM Journal on Computing 40, 63–78 (2011)
17. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. Journal of the ACM 46(2), 858–923 (1999)
18. Moir, M., Anderson, J.H.: Fast, Long-Lived Renaming (Extended Abstract). In: Tel, G., Vitányi, P.M.B. (eds.) WDAG 1994. LNCS, vol. 857, pp. 141–155. Springer, Heidelberg (1994)
19. Okun, M.: Strong order-preserving renaming in the synchronous message passing model. Theoretical Computer Science 411(40-42), 3787–3794 (2010)
20. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (1980)

On Snapshots and Stable Properties Detection in Anonymous Fully Distributed Systems (Extended Abstract)

Jérémie Chalopin[1], Yves Métivier[2], and Thomas Morsellino[2]

[1] LIF, CNRS & Aix-Marseille Université
39, rue Joliot Curie
13453 Marseille Cedex 13, France
jeremie.chalopin@lif.univ-mrs.fr
[2] Université de Bordeaux, LaBRI, UMR CNRS 5800
351, cours de la Libération
33405 Talence, France
{metivier,morsellino}@labri.fr

Abstract. Most known snapshot algorithms assume that the vertices of the network have unique identifiers and/or that there is exactly one initiator. This paper concerns snapshot computation in an anonymous network and more generally what stable properties of a distributed system can be computed anonymously with local snapshots with multiple initiators when knowing an upper bound on the diameter of the network.

1 Introduction

The Problem. A distributed system (P, C) consists of a collection P of processes and a communication subsystem C. It is described by a simple connected undirected graph $G = (V, E)$, where the vertices represent the processes and the edges represent the bidirectional channels.

A message passing algorithm is defined as follows: to each process is associated a state and a transition system which can modify the state of the process and which can interact with the communication subsystem. The events which are associated with a process are internal events, send events and receive events. In a send (resp. receive) event a message is produced (resp. consumed).

Let Q be the (recursive) set of possible states of p. Let $\mathcal{M}$ be the set of possible messages. The state of a channel is the multiset of messages sent through this channel and not yet received. Let p be a process, the *local snapshot* with respect to p is defined by the state of p and by the states of incoming channels to p.

The state of a distributed system G, also called its *global state,* is defined by the state of each process and the state of each channel (or equivalently by the set of local snapshots): it is precisely a *(global) snapshot.* Thus a snapshot of a distributed system is an instantaneous map of it, where each vertex (resp. each edge) is labelled by its state. It will be denoted by $\mathbf{G} = (G, \lambda)$ where λ is a labelling function which associates to each vertex (resp. each edge) its state.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 207–218, 2012.

By definition, in a fully distributed asynchronous system there is no global clock and no process has the knowledge of a snapshot; each processor knows, a priori, only its state. It knows neither the states of the other processors nor the state of any channel.

Given a distributed system, the aim of a snapshot algorithm is the computation of such a global state.

As explained by Tel [Tel00] (p. 335-336), the construction of snapshots can be useful to detect stable properties of the distributed system (properties which remain true as soon as they are verified), to restart the system from the last known snapshot (and not from the initial configuration) when the system must be restarted (due to a failure of a component), or to debug distributed algorithms.

A *consistent snapshot* of a distributed system is a global state of the distributed system or a global state that the system could have reached. Since the seminal paper of Chandy and Lamport [CL85] which presents an algorithm to compute a consistent snapshot, many papers give such algorithms according to the model of the distributed system. They assume that processes have unique identifiers and/or that there is exactly one initiator. Many papers give also specific algorithms to detect some specific properties like termination or deadlock.

Recently, Guerraoui and Ruppert [GR05], considering that a vast majority of papers on distributed computing assume that processes have unique identifiers, ask the following question: *What if processes do not have unique identifiers or do not wish to divulge them for reasons of privacy?*

In this paper, we consider this question in the context of snapshots computations and by considering stable properties of a distributed system that can be detected anonymously. Furthermore, we look for fully distributed solutions which admit several initiators.

The Model. As we said before, our model is the usual asynchronous message passing model ([Tel00,YK96b]). A network is represented by a simple connected graph $G = (V(G), E(G)) = (V, E)$ where vertices correspond to processes and edges to direct communication links. The state of each process (resp. each link e) is represented by a label $\lambda(v)$ (resp. $\lambda(e)$) associated to the corresponding vertex $v \in V(G)$ (resp. link $e \in E$); we denote by $\mathbf{G} = (G, \lambda)$ such a labelled graph. We assume the network to be anonymous: the identities of processors are not necessarily unique.

We assume that each process can distinguish the different edges that are incident to it, i.e., for each $u \in V(G)$ there exists a bijection δ_u between the neighbors of u in G and $[1, \deg_G(u)]$. We will denote by δ the set of functions $\{\delta_u \mid u \in V(G)\}$. The numbers associated by each vertex to its neighbors are called *port-numbers* and δ is called a *port-numbering* of G. We will denote by $(\mathbf{G}, \delta)$ the labelled graph $\mathbf{G}$ with the port-numbering δ.

Each process v in the network represents an entity that is capable of performing computation steps, sending messages via some port and receiving any message via some port that was sent by the corresponding neighbor. We consider asynchronous systems, i.e., no global time is available and each computation may take an unpredictable (but finite) amount of time. Note that we consider only

reliable systems: no fault can occur on processes or communication links. We also assume that the channels are FIFO, i.e., for each channel, the messages are delivered in the order they have been sent. In this model, a distributed algorithm is given by a local algorithm that all processes should execute. A local algorithm consists of a sequence of computation steps interspersed with instructions to send and to receive messages. We follow the presentation and definitions given in [Tel00] (p. 45-47) or [AW04] (p. 10-12).

Our Contribution. We assume that the network is anonymous and that several processes can be initiators of computations thus no process of the network can compute a snapshot, i.e., can know a map of the network with vertices and edges labelled by states of processes and channels (it is a direct consequence of Theorem 5.5 in [Ang80]). Furthermore we assume that each process knows an upper bound of the diameter of the network.

First we give a very simple algorithm based on the composition of an algorithm by Szymanski, Shy, and Prywes [SSP85] with the Chandy-Lamport algorithm which enables each process: to detect an instant where all processes have obtained their local snapshot, and to associate the same number to all local snapshots. By this way we obtain two applications: one to checkpoint and rollback recovery and a second to detect termination of the execution of a distributed algorithm.

Then we prove that some stable properties can be anonymously detected by proving we can compute a snapshot up to covering (called a weak snapshot). In some sense, the weak snapshot is the "global view" or the "maximal knowledge" of the distributed system that each vertex can obtain anonymously.

Related Work. Many notions and algorithms concerning snapshots, stable properties, checkpointing and rollback recovery can be found in [KS08].

From a theoretical point of view, it is simple to know whether the global state of a distributed system satisfies a stable property. A distinguished process starts the Chandy-Lamport algorithm, then it collects states of processes and states of channels, it computes a map of the network and finally it tests whether the labelled network satisfies the given property.

To collect or to analyze local snapshots, different assumptions may be done (see [KRS95]): processes have unique identifiers, there is exactly one initiator or one collector process.

Some results have been obtained for the computation of snapshots in asynchronous shared-memory systems that are anonymous : [GR05] (Section 5) gives a survey. This paper also presents results concerning consensus and timestamping.

The question "What can be computed anonymously?" has been explored in the asynchronous message passing model for the election problem, symmetry breaking and more generally for computing functions [Ang80, BCG+96, JS85, YK96b, YK96a, YK99, BV99]. Angluin has introduced the classical proof techniques used for showing impossibilities based on coverings.

2 The Chandy-Lamport Snapshot Algorithm

The aim of a snapshot algorithm is to construct a system configuration defined by the state of each process and the state of each channel.

This section presents the Chandy-Lamport snapshot algorithm [CL85]; it is presented as Algorithm 1. Each process p is equipped with: a boolean variable $taken_p$ which is initialized to $false$ that indicates if the process p has already recorded its state, a boolean variable $local\text{-}snapshot_p$ initialized to $false$ that indicates if the process p has recorded its state and the state of incoming channels, and a multiset of messages $M_{p,i}$, initially $M_{p,i} = \emptyset$, for each incoming channel i of p.

We assume that Algorithm 1 is initiated by at least one process which: saves its state, sends a marker on each outcoming port and for each incoming port memorizes messages which arrive until it receives a marker through this port. When a process receives for the first time a marker, it does the same thing that an initiator; the incoming channel by which it receives for the first time a marker is saved as empty.

I$nit\text{-}CL_p$: {To initiate the algorithm by at least one process p such that $taken_p = false$}
begin
  record(state(p)) ;
  $taken_p := true$;
  send$< mkr >$ to each neighbor of p;
  For each port i the process p records messages arriving via i
end

R-CL_p : {A marker has arrived at p via port j}
begin
  receive$< mkr >$;
  mark port j;
  if *not* $taken_p$ **then**
    $taken_p := true$;
    record(state(p)) ;
    send$< mkr >$ via each port;
    For each port $i \neq j$ the process p records messages arriving via i in $M_{p,i}$
  else
    The process p stops to record messages from the channel j of p;
    record($M_{p,j}$)
  if p *has received a marker via all incoming channels* **then**
    $local\text{-}snapshot_p := true$
end

Algorithm 1. The Chandy-Lamport snapshot algorithm

If we consider an execution of the Chandy-Lamport algorithm we obtain a consistent snapshot within finite time after its initialization by at least one process (see [Tel00] Theorem 10.7). In particular:

Fact 1. *Within finite time after the initialization of the Chandy-Lamport algorithm, each process p has computed its local snapshot ($local\text{-}snapshot_p = true$).*

Once the computation of local snapshots is completed (for each process p the boolean $local\text{-}snapshot_p$ is true), the knowledge of the snapshot is fully

distributed over the system. The next question is "how to exploit this distributed knowledge?".

A first answer is obtained by the construction of the global state of the system centralized on a process. As is explained by Raynal [Ray88]: *Providing an algorithm for the calculation of a global state is a basic problem in distributed systems.* Several assumptions can be done to obtain a global state: exactly one initiator for the Chandy-Lamport algorithm, processes have unique identifiers or global colors associated to each computation of a global state...

A global clock can be simulated by local logical clocks [Ray88], nevertheless it does not enable iterated computations of snapshots.

Another way to exploit the local knowledge is based on wave algorithms: a message is passed to each process by a single initiator following the topology of the network or a virtual topology (ring, tree, complete graph, ...), see [MC98].

These solutions are not available in the context of anonymous networks with no distinguished process and no particular topology.

3 Termination Detection of the Chandy-Lamport Snapshot Algorithm

A first problem is the termination detection of the computation of all local snapshots. It requires that all vertices certify, in a finite computation, that they have completed the computation of the local snapshot. The algorithm by Szymanski, Shy, and Prywes (the SSP algorithm for short) [SSP85] does this for a region of pre-specified diameter. The algorithm assumes that an upper bound of the diameter of the entire network is known by each process. In the sequel this upper bound is denoted by β and we assume that each process knows it. Now, we present the SSP algorithm we use in this paper.

The SSP Algorithm. We consider a distributed algorithm which terminates when all processes reach their local termination conditions. Each process is able to determine only its own termination condition. SSP's algorithm detects an instant in which the entire computation is achieved.

Let G be a graph, to each process p is associated a predicate $P(p)$ and an integer $a(p)$. Initially $P(p)$ is false and $a(p)$ is equal to -1. Transformations of the value of $a(p)$ are defined by the following rules.

Each local computation acts on the integer $a(p_0)$ associated to the process p_0; the new value of $a(p_0)$ depends on values associated to neighbors of p_0. More precisely, let p_0 be a process and let $\{p_1, ..., p_d\}$ the set of processes adjacent to p_0. If $P(p_0) = false$ then $a(p_0) = -1$; if $P(p_0) = true$ then $a(p_0) = 1 + Min\{a(p_k) \mid 1 \leq k \leq d\}$. We assume that for each process p the value of $P(p)$ eventually becomes true and remains true for ever. To apply the SSP algorithm, the label of each process has two items:

- $a(p) \in \mathbb{Z}$ is a counter and initially $a(p) = -1$, $a(p)$ represents the distance up to which all processes have the predicate true;

– $A(p) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{Z})$[1] encodes the information p has about the values of $a(q)$ for each neighbor q. Initially, $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$.

We consider an execution $\mathcal{E}$ of the SSP's algorithm on the graph $\mathbf{G}$ and $(a_i(p))_{i \geq 0}$ the sequence defined by the values of $a(p)$ for the execution $\mathcal{E}$. The predicate P is true for each process of the ball of radius $a_i(p)$ with center p, i.e.:

Proposition 1. *Let p be a process of G, we suppose that $h = a_i(p) \geq 0$. Then for each $q \in V(G), d(p, q) \leq h \Rightarrow a_i(q) \geq 0$.*

Thus a process p such that $a(p)$ is greater or equal than the diameter of the graph knows that for each process q of the graph $P(q)$ is true, i.e., it detects the termination of the algorithm.

3.1 An Algorithm to Detect Termination of the Local Snapshots Computation

Now, we compose the application of the Chandy-Lamport algorithm and the SSP algorithm to enable each process to detect an instant where all processes have completed the computation of their local snapshot: it is given as Algorithm 2.

Since we want the algorithm to be able to compute snapshots at different times in the execution, we add a variable $\textit{snapshot-number}_p$ which indicates the number of the snapshot (initially, $\textit{snapshot-number}_p = 0$). This variable is not really necessary in this section, but it will be used in the sequel.

Here, a process starts executing the SSP algorithm once it has computed its pth local snapshot, i.e., when $\textit{local-snapshot}_p$ is true.

A process p knows that each process has completed the computation of its local snapshot as soon as $a(p) \geq \beta$ (we recall that β is an upper bound of the diameter of the network). Thus we add a boolean variable *snapshot* initialized to false; it indicates if the process knows whether all processes have completed the computation of the local snapshots.

Proposition 2. *let (G, λ) be a network. Let β be an upper bound of the diameter of G known by each process. Within finite time after the initialization of the Chandy-Lamport algorithm, Algorithm 2 enables each process to know an instant where all processes have completed the computation of their local snapshots.*

4 Two Applications: "Checkpoint and Rollback Recovery" and "Termination Detection"

This section presents two simple applications of the Chandy-Lamport algorithm and of Algorithm 2 in the anonymous context, without unicity of an initiator and assuming that each process knows an upper bound of the diameter of the network: 1. to compute configurations to restart the system when a process

[1] For any set S, $\mathcal{P}_{\text{fin}}(S)$ denotes the set of finite subsets of S.

```
Init-SSP_p : {To initiate termination detection on the process p such that local-snapshot_p =
true, a(p) = −1 and snapshot = false}
begin
    a(p) := 0;
    m := Min{x | (i, x) ∈ A(p)};
    if m ≥ a(p) then
        a(p) := m + 1 ;
    send< a(p) > to each neighbor of p
end
R-SSP_p : {An integer < α > has arrived at p via port j}
begin
    receive< α >;
    A(p) := (A(p) \ {(j, x)}) ∪ {(j, α)};
    m := Min{x | (i, x) ∈ A(p)};
    if (m ≥ a(p) and local-snapshot_p = true) then
        a(p) := m + 1;
    if a(p) ≥ β then
        snapshot-number_p := snapshot-number_p + 1;
        snapshot_p := true
    else
        send< a(p) > via each port
end
```

Algorithm 2. Termination detection of the Chandy-Lamport snapshot algorithm

fails (see [KS08] Chapter 13 for a presentation of checkpointing and rollback recovery), 2. to detect the termination of an execution of a distributed algorithm.

An Application to Checkpoint and Rollback Recovery. A snapshot enables to restart a system if there is a failure. As explained in [KS08] p. 456, *the saved state is called a checkpoint, and the procedure of restarting from a previously checkpointed state is called rollback recovery.*

Our solution is obtained by repeatedly executing the following steps:

1. at least one process initiates the Chandy-Lamport algorithm (Algorithm 1);
2. each process p detects an instant where the computation of its local snapshot is completed: $\textit{local-snapshot}_p = \textit{true}$;
3. each process p detects an instant where the computation of all local snapshots is completed: $\textit{snapshot}_p = \textit{true}$ (Algorithm 2);
4. a new number (obtained by adding 1 to the counter $\textit{snapshot-number}_p$, initially $\textit{snapshot-number}_p = 0$) is associated to this snapshot and each process p gives this number to its local snapshot. Each process p saves its last local snapshot associated to the number $\textit{snapshot-number}_p$. It enables to restart if there is a failure.
5. Finally, variables for Algorithm 1 and Algorithm 2 are reset at the end.

From Local Snapshots Computation to Termination Detection of the Execution of a Distributed Algorithm

Let $\mathcal{A}$ be a distributed algorithm. Let $\mathcal{E}$ be an execution of $\mathcal{A}$. Our aim is to detect the termination of $\mathcal{E}$.

An execution $\mathcal{E}$ has terminated if and only if all the processes are passive and all the channels are empty. Thus to detect the termination of the execution $\mathcal{E}$, it

suffices that from time to time (to be defined) at least one process initializes the computation of a snapshot and if its state is passive and its incoming channels are empty it must detect if the same property holds for all the processes. This is done by using an occurence of the SSP algorithm. If variables of a process p indicate that the execution is not completed then q emits a signal through the network to inform each process.

In this way, we obtain an algorithm to detect global termination of the execution of a distributed algorithm. These repeated termination queries are analogue to the solution described by Santoro in Section 8.3 of [San07].

5 Coverings, Stable Properties and Weak Snapshots

We assume that the network is anonymous and that several processes can be initiators of computations. Each process knows only an upper bound of the diameter, denoted β. Under these hypotheses, no process can compute a map of the network. We prove that each process can compute a graph covered by the network, i.e., a weak snapshot. We prove also that classical properties, as stable properties, studied through snapshots can be still studied thanks to a weak snapshot.

In the following, we will consider directed graphs (digraphs) with multiple arcs and self-loops. A *digraph* $D = (V(D), A(D), s_D, t_D)$ is defined by a set $V(D)$ of vertices, a set $A(D)$ of arcs and by two maps s_D and t_D that assign to each arc two elements of $V(D)$: a source and a target (in general, the subscripts will be omitted). A *symmetric* digraph D is a digraph endowed with a symmetry, that is, an involution $Sym : A(D) \to A(D)$ such that for every $a \in A(D), s(a) = t(Sym(a))$. In a symmetric digraph D, the degree of a vertex v is $\deg_D(v) = |\{a \mid s(a) = v\}| = |\{a \mid t(a) = v\}|$. Let (G, λ) be a labelled graph with the port-numbering δ. We will denote by $(\mathrm{Dir}(\mathbf{G}), \delta)$ the symmetric labelled digraph $(\mathrm{Dir}(G), (\lambda, \delta))$ constructed in the following way. The vertices of $\mathrm{Dir}(G)$ are the vertices of G and they have the same labels in $\mathbf{G}$ and in $\mathrm{Dir}(\mathbf{G})$. Each edge $\{u, v\}$ of G is replaced in $(\mathrm{Dir}(\mathbf{G}), \delta)$ by two arcs $a_{(u,v)}, a_{(v,u)} \in A(\mathrm{Dir}(G))$ such that $s(a_{(u,v)}) = t(a_{(v,u)}) = u$, $t(a_{(u,v)}) = s(a_{(v,u)}) = v$, $\delta(a_{(u,v)}) = (\delta_u(v), \delta_v(u))$ and $\delta(a_{(v,u)}) = (\delta_v(u), \delta_u(v))$. Note that this digraph does not contain multiple arcs or loop.

The notion of coverings and of symmetric coverings are fundamental in this work; definitions and main properties are presented in [GT87,BV02]. This notion enables to express "similarity" between two digraphs. A labelled digraph $\mathbf{D}$ is a *covering* of a labelled digraph $\mathbf{D'}$ via φ if φ is a homomorphism from $\mathbf{D}$ to $\mathbf{D'}$ such that for each arc $a' \in A(D')$ and for each vertex $v \in \varphi^{-1}(t(a'))$ (resp. $v \in \varphi^{-1}(s(a'))$, there exists a unique arc $a \in A(D)$ such that $t(a) = v$ (resp. $s(a) = v$) and $\varphi(a) = a'$. A symmetric labelled digraph $\mathbf{D}$ is a *symmetric covering* of a symmetric labelled digraph $\mathbf{D'}$ via φ if $\mathbf{D}$ is a covering of $\mathbf{D'}$ via φ and if for each arc $a \in A(D)$, $\varphi(Sym(a)) = Sym(\varphi(a))$. The homomorphism φ is a *symmetric covering projection* from $\mathbf{D}$ to $\mathbf{D'}$. Given a simple connected labelled graph $\mathbf{G} = (G, \lambda)$ with a port-numbering δ which defines a snapshot of a network

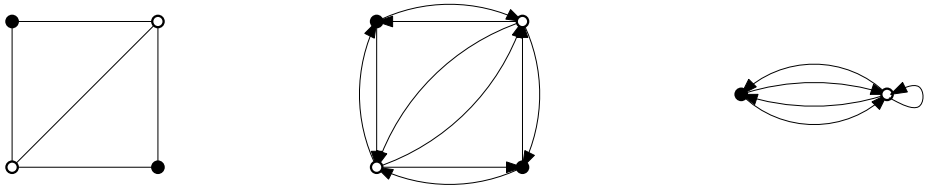


Fig. 1. A graph G, the corresponding digraph $Dir(G)$ and a digraph D' such that $Dir(G)$ is a symmetric covering of D'

G. Let $\mathbf{D} = (\mathrm{Dir}(\mathbf{G}), \delta)$ be the corresponding labelled digraph $(\mathrm{Dir}(G), (\lambda, \delta))$. Let $\mathbf{D}'$ be a labelled digraph such that $\mathbf{D} = (\mathrm{Dir}(\mathbf{G}), \delta)$ is a covering of $\mathbf{D}'$. The labelled digraph $\mathbf{D}'$ is called a weak snapshot of G.

Let $\mathbf{G}$ be a graph. Let $\mathcal{D}$ be a message passing algorithm and let $\mathcal{E}$ be an execution of $\mathcal{D}$ over $\mathbf{G}$. A property P of configurations of $\mathcal{E}$ is stable: if P is true for a configuration $(\mathtt{state}, M)$ then P is true for any configuration obtained from $(\mathtt{state}, M)$. Among stable properties of distributed systems detected with snapshot, we consider: termination, deadlock, loss of tokens and garbage collection (see [Tel00, San07, KS08]).

Termination: An execution $\mathcal{E}$ has terminated if and only if all processes are passive and channels are empty. The link of this property with the computation of a snapshot has been treated in Section 4.

Deadlock: A deadlock happens in a distributed system if there is a cycle of processes each waiting for the next in the cycle with no message in transit. It can be detected by constructing the *wait-for-graph* (WFG for short): vertices are processes and there is an arc from the vertex (process) p to the vertex q if p is blocked and is waiting for q. There is a deadlock if and only if there exists a cycle in the WFG (see [San07, KS08]).

Loss of Tokens: Some distributed systems need that tokens circulate among processes. In order to check if the number of tokens existing in the system is correct, it may be interesting to check properties like "there are exactly k tokens" or "there are at most k tokens"; these properties are stable.

Garbage Collection: The aim is to decide if an object is useful.We follow the presentation of Schiper and Sandoz [SS94]: Consider a system composed of a set of objects O, and a static subset $Root \subseteq O$, called root objects. Root objects are invoked by some processes. An invocation on object o_i implies the execution of actions by o_i. The set of objects can be represented as a set of processes exchanging messages upon invocation, and messages can carry references. On the set O of objects, define the descendant relation by: descendent (o_i, o_j) if object o_i holds a reference on object o_j or a reference on o_j is under way to o_i. An object o_i is reachable either if $o_i \in Root$ or if o_i is descendant of a reachable object. By definition an object is useful if it is reachable. Only reachable object

can send references to other objects. An object that is no more reachable in the system is called garbage and should be destroyed. Thus reachable objects are detected as vertices o for which there is a walk from a root to o.

Stable Properties and Snapshots. Usually, after the computation of a snapshot, stable properties are verified by using a spanning tree or an embedded ring; they also may be verified in a centralized way by computing on a vertex a map of the network each vertex and each channel being labelled with its state.

These solutions are no longer possible in an anonymous network with no distinguished vertex. In this context, the tool we use is covering.

Stable Properties in Distributed Systems and Weak Snapshot. Let $\mathbf{D}_1$ and $\mathbf{D}_2$ be two labelled digraphs such that $\mathbf{D}_1$ is a covering of $\mathbf{D}_2$ via the homorphism φ. A lift of a walk W_2 in $\mathbf{D}_2$ is a walk W_1 in $\mathbf{D}_1$ such that φ maps the arcs of W_1 onto the arcs of W_2 in the order of traversal. Let $(\mathbf{G}, \delta)$ be a network with a port numbering δ. Let $\mathbf{D}'$ be a labelled graph such that $(Dir(\mathbf{G}), \delta)$ is a covering of $\mathbf{D}'$. With our notations, Theorem 2.4.1 and Theorem 2.4.3 in [GT87] can be translated by:

1. Let W be a walk in $\mathbf{D}'$ such that the initial vertex is u. Then for each u_i in $\varphi^{-1}(u)$ there is a unique lift of W that starts at u_i.
2. Let C be a cycle in $\mathbf{D}'$. Then $\varphi^{-1}(C)$ is an union of cycles.

From these two results, we deduce that if $\mathbf{D}_1$ is a covering of $\mathbf{D}_2$ then deadlock and garbage detected in $\mathbf{D}_1$ can be detected in $\mathbf{D}_2$. We recall that if $\mathbf{D}_1$ is a covering of $\mathbf{D}_2$ via the homorphism φ then there exists an integer α such that for each vertex u of $\mathbf{D}_2$ the cardinality of $\varphi^{-1}(u)$ is equal to α. Thus if there are c_1 tokens in $\mathbf{D}_1$ (for a non negative integer c_1) then there are $c_2 = c_1/\alpha$ tokens in $\mathbf{D}_2$. From this, we deduce that if the size of $\mathbf{D}_1$ is known then the knowledge of $\mathbf{D}_2$ enables to detect loss of tokens in $\mathbf{D}_2$. These facts are summarized by:

Proposition 3. *Let G be a distributed system. From any weak snapshot $\mathbf{D}$ of G, one can detect deadlock and termination and one can perform garbage collection. Furthermore, if the processes know the size of G then loss of tokens can also be detected from a weak snapshot.*

In the following, we describe a fully distributed algorithm (it may admit several initiators) with termination detection, which computes $\mathbf{D}_2$ such that $\mathbf{D}_1$ is a covering of $\mathbf{D}_2$. Using such an algorithm, from Proposition 3, we can solve stable properties detection. It suffices that:

1. at least one process initiates the Chandy-Lamport algorithm (Algorithm 1);
2. each process detects an instant where the computation of all local snapshots is completed (Algorithm 2);
3. at least one process initiates the computation of a weak snapshot;
4. each process detects an instant where the computation of the weak snapshot is completed and decides about the stable property.

Computing Anonymously a Weak Snapshot. Let $(\mathbf{G}, \delta)$ be a labelled graph with a port numbering δ. We assume that $\mathbf{G}$ is anonymous and each vertex of

G knows an upper bound, denoted β, of the diameter of G. There exists an algorithm, denoted $\mathcal{M}_{W-S}$, which computes a weak snapshot, i.e., a labelled digraph $(\mathbf{D}, \delta')$ such that $(Dir(\mathbf{G}), \delta)$ is a covering of $(\mathbf{D}, \delta')$. In some sense, the weak snapshot $(\mathbf{D}, \delta')$ is the "global view" or the "maximal knowledge" of the distributed system that each vertex can obtain ([Ang80], Theorem 5.5). If processes know an upper bound of the diameter of the network then they can detect the termination of $\mathcal{M}_{W-S}$. This algorithm has been presented in [Cha06,CM07] as an election algorithm when it is executed on minimal graphs for the symmetric-covering relation (the graph $(\mathbf{G}, \delta)$ is minimal if when $(Dir(\mathbf{G}), \delta)$ is a symmetric-covering of $(\mathbf{D}, \delta')$ then $(Dir(\mathbf{G}), \delta)$ is isomorphic to $(\mathbf{D}, \delta')$). This algorithm is based on another election algorithm given by Mazurkiewicz in [Maz97] and the SSP algorithm. During the execution of the algorithm, each vertex v attempts to get an identity which is a number between 1 and $|V(G)|$. Once a vertex v has chosen a number $n(v)$, it sends it to each neighbor u with the port-number $\delta_v(u)$. When a vertex u receives a message from one neighbor v, it stores the number $n(v)$ with the port-numbers $\delta_u(v)$ and $\delta_v(u)$. From all information it has gathered from its neighbors, each vertex can construct its *local view* (which is the set of numbers of its neighbors associated with the corresponding port-numbers). Then, a vertex broadcasts its number, its label and its mailbox (which contains a set of *local views*). If a vertex u discovers the existence of another vertex v with the same number then it should decide if it changes its identity. To this end it compares its local view with the local view of v. If the label of u or the local view of u is "weaker", then u picks another number — its new temporary identity — and broadcasts it again with its local view. At the end of the computation, each vertex has computed a graph $(\mathbf{D}, \delta')$ such that $(Dir(\mathbf{G}), \delta)$ is a symmetric covering of $(\mathbf{D}, \delta')$.

References

[Ang80] Angluin, D.: Local and global properties in networks of processors. In: Proceedings of the 12th Symposium on Theory of Computing, pp. 82–93 (1980)

[AW04] Attiya, H., Welch, J.: Distributed computing: fundamentals, simulations, and advanced topics. John Wiley & Sons (2004)

[BCG+96] Boldi, P., Codenotti, B., Gemmell, P., Shammah, S., Simon, J., Vigna, S.: Symmetry breaking in anonymous networks: Characterizations. In: Proc. 4th Israeli Symposium on Theory of Computing and Systems, pp. 16–26. IEEE Press (1996)

[BV99] Boldi, P., Vigna, S.: Computing anonymously with arbitrary knowledge. In: Proceedings of the 18th ACM Symposium on Principles of Distributed Computing, pp. 181–188. ACM Press (1999)

[BV02] Boldi, P., Vigna, S.: Fibrations of graphs. Discrete Math. 243, 21–66 (2002)

[Cha06] Chalopin, J.: Algorithmique distribuée, calculs locaux et homorphismes de graphes. PhD thesis, université Bordeaux 1 (2006)

[CL85] Chandy, K.M., Lamport, L.: Distributed snapshots: Determining global states of distributed systems. ACM Trans. Comput. Syst. 3(1), 63–75 (1985)

[CM07] Chalopin, J., Métivier, Y.: An efficient message passing election algorithm based on mazurkiewicz's algorithm. Fundam. Inform. 80(1-3), 221–246 (2007)

[GR05] Guerraoui, R., Ruppert, E.: What Can Be Implemented Anonymously? In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 244–259. Springer, Heidelberg (2005)

[GT87] Gross, J.L., Tucker, T.W.: Topological graph theory. Wiley Interscience (1987)

[JS85] Johnson, R.E., Schneider, F.B.: Symmetry and similarities in distributed systems. In: Proc. 4th Conf. on Principles of Distributed Computing, pp. 13–22 (1985)

[KRS95] Kshemkalyani, A.D., Raynal, M., Singhal, M.: An introduction to snapshot algorithms in distributed computing. Distributed Systems Engineering 2(4), 224–233 (1995)

[KS08] Kshemkalyani, A.D., Singhal, M.: Distributed computing, Cambridge (2008)

[Maz97] Mazurkiewicz, A.: Distributed enumeration. Inf. Processing Letters 61, 233–239 (1997)

[MC98] Matocha, J., Camp, T.: A taxonomy of distributed termination detection algorithms. Journal of Systems and Software 43(3), 207–221 (1998)

[Ray88] Raynal, M.: Networks and distributed computation. MIT Press (1988)

[San07] Santoro, N.: Design and analysis of distributed algorithm. Wiley (2007)

[SS94] Schiper, A., Sandoz, A.: Strong stable properties in distributed systems. Distributed Computing 8(2), 93–103 (1994)

[SSP85] Szymanski, B., Shy, Y., Prywes, N.: Synchronized distributed termination. IEEE Transactions on Software Engineering SE-11(10), 1136–1140 (1985)

[Tel00] Tel, G.: Introduction to distributed algorithms. Cambridge University Press (2000)

[YK96a] Yamashita, M., Kameda, T.: Computing functions on asynchronous anonymous networks. Math. Systems Theory 29, 331–356 (1996)

[YK96b] Yamashita, M., Kameda, T.: Computing on anonymous networks: Part i - characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems 7(1), 69–89 (1996)

[YK99] Yamashita, M., Kameda, T.: Leader election problem on networks in which processor identity numbers are not distinct. IEEE Transactions on Parallel and Distributed Systems 10(9), 878–887 (1999)

Self-stabilizing (k,r)-Clustering in Clock Rate-Limited Systems

Andreas Larsson and Philippas Tsigas

Chalmers University of Technology and Göteborg University
{larandr,tsigas}@chalmers.se

Abstract. Wireless Ad-hoc networks are distributed systems that often reside in error-prone environments. Self-stabilization lets the system recover autonomously from an arbitrary system state, making the system recover from errors and temporarily broken assumptions. Clustering nodes within ad-hoc networks can help forming backbones, facilitating routing, improving scaling, aggregating information, saving power and much more. We present a self-stabilizing distributed (k,r)-clustering algorithm. A (k,r)-clustering assigns k cluster heads within r communication hops for all nodes in the network while trying to minimize the total number of cluster heads. The algorithm assumes a bound on clock frequency differences and a limited guarantee on message delivery. It uses multiple paths to different cluster heads for improved security, availability and fault tolerance. The algorithm assigns, when possible, at least k cluster heads to each node within $O(r\pi\lambda^3)$ time from an arbitrary system configuration, where π is a limit on message loss and λ is a limit on pulse rate differences. The set of cluster heads stabilizes, with high probability, to a local minimum within $O(r\pi\lambda^4 g \log n)$ time, where n is the size of the network and g is an upper bound on the number of nodes within $2r$ hops.

1 Introduction

Starting from an arbitrary system state, self stabilizing algorithms let a system stabilize to, and stay in, a consistent system state [5]. There are many reasons why a system could end up in an inconsistent system state of some kind. Assumptions that algorithms rely on could temporarily be invalid. Memory content could be changed by radiation or other elements of harsh environments. Battery powered nodes could run out of batteries and new ones could be added to the network. It is often not feasible to manually configure large ad-hoc networks to recover from events like this. Self-stabilization is therefore often a desirable property of algorithms for ad-hoc networks. However, the trade off is that self-stabilization often comes with increased costs. A self-stabilizing algorithm can never stop because it is not known in advance when temporary faults occur. Nevertheless, as long as all assumptions hold, it can converge to stable result, or, after convergence, stay within a set of acceptable states. Moreover, there are

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 219–230, 2012.

often overheads in the algorithm tied to the need to recover from arbitrary system states. They can be additional computations, larger messages, larger data structures or longer required times to achieve certain goals.

An algorithm for clustering nodes together in an ad-hoc network serves an important role. Back bones for efficient communication can be formed using cluster heads. Clusters can be used for routing messages. Cluster heads can be responsible for aggregating data into reports to decrease the number of individual messages that needs to be routed through the network, e.g., aggregating sensor readings in a wireless sensor network. Hierarchies of clusters on different levels can be used for improved scaling of a large network. Nodes in a cluster could take turns doing energy-costly tasks to reduce overall power consumption.

Clustering is a well studied problem. Due to space constraints, for references to the area in general, we point to the survey of the area with regard to wireless ad-hoc networks by Chen, Liestam and Liu in [4] and the survey by Abbasi and Younis in [1] for wireless sensor networks. In this paper we focus on self-stabilization, redundancy and security aspects. One way of clustering nodes in a network is for nodes to associate themselves with one or more cluster heads. In the (k,r)-clustering problem each node in the network should have at least k cluster heads within r communication hops away. This might not be possible for all nodes if the number of nodes within r hop from them is smaller than k. In such cases a best effort approach can be taken for getting as close to k cluster heads as possible for those nodes. The clustering should be achieved with as few cluster heads as possible. To find the global minimum number of cluster heads is in general computationally hard, and algorithms usually provide approximations. The (1,r)-clustering problem, a subset of the (k,r)-clustering problem, can be formulated as a classical set cover problem. This was shown to be NP complete in [10]. Assuming that the network allows k cluster heads for each node, the set of cluster heads forms a total (k,r)-dominating set in the network. In a *total* (k,r)-dominating set the nodes in the set also need to have k nodes in the set within r hops, in contrast to an ordinary (k,r)-dominating set in which this is only required for nodes not in the set.

There is a multitude of existing clustering algorithms for ad-hoc networks of which a number is self-stabilizing. Johnen and Nguyen present a self-stabilizing (1,1)-clustering algorithm that converges fast in [9]. Dolev and Tzachar tackle a lot of organizational problems in a self-stabilizing manner in [6]. As part of this work they present a self-stabilizing (1,r)-clustering algorithm. Caron, Datta, Depardon and Larmore present a self-stabilizing (1,r)-clustering in [3] that takes weighted graphs into account. Self-stabilization in systems with unreliable communications was introduced in [2]. In [14] a self-stabilizing (k,1)-clustering algorithm, that can cope with message loss, is presented. There is a number of papers that do not have self-stabilization in their settings. Fu, Wang and Li consider the (k,1)-clustering problem in [7]. In [15] the full (k,r)-clustering problem is considered and both a centralized and a distributed algorithm for solving this problem are presented. Wu and Li also consider the full (k,r)-clustering in [17]. Other algorithms do not take the cluster head approach. In [16], sets of nodes

that all can communicate directly with each other are grouped together without assigning any cluster heads. In this paper malicious nodes that try to disturb the protocol are also considered, but self-stabilization is not considered.

We have constructed a self-stabilizing (k, r)-clustering algorithm for ad-hoc networks that can deal with message loss, as long as at least one out of π consecutive broadcasts are successful, and that uses unsynchronized pulses, for which the ratios between pulse rates are limited by a factor λ. The algorithm makes sure that, within $O(r\pi\lambda^3)$ time, all nodes have at least k cluster heads (or all nodes within r hops if a node has less than k nodes within r hops) using a deterministic scheme. A randomized scheme complements the deterministic scheme and lets the set of cluster heads stabilize to a local minimum. It stabilizes within $O(r\pi\lambda^4 g \log n)$ time with high probability, where g is a bound on the number of nodes within $2r$ hops, and n is the size of the network. We presented the first distributed self-stabilizing (k,r)-clustering in [11]. There, the system settings assumed perfect message transfers and lock step synchronization of the nodes. The current article is a further development of that work and the main idea of the algorithm is the same. The unreliable communication media, the unsynchronized nodes and the introduction of a veto mechanism to speed up convergence, all have made the current algorithm quite different, yet clearly related to the one in [11]. We present an overview of our correctness proofs for quick selection of enough cluster heads (k cluster heads within r hops when possible) and that the set of cluster heads converges towards a local minimum and stays. This includes an upper bound on the time it takes, with high probability, for that convergence to happen. Due to space constraints we refer the reader to [12] for the full set of proofs. Furthermore, we also present experimental results on the convergence of the algorithm and how it copes with changes to the topology. The rest of the paper is organized as follows. In section 2 we introduce the system settings. Section 3 describes the algorithm. Section 4 gives the overview of the proofs of the algorithm. We discuss experimental results, security and redundancy in Section 5.

2 System Settings

We assume a static network. Changes in the topology are seen as transient faults. We denote the set of all nodes in the network $\mathcal{P}$ and the size of the network $n = |\mathcal{P}|$. We impose no restrictions on the network topology other than that an upper bound, g, on the number of nodes within $2r$ hops of any node is known (see below).

The set of neighbors, N_i, of a node p_i is all the nodes that can communicate directly with node p_i. In other words, a node $p_j \in N_i$ is one hop from node p_i. We assume a bidirectional connection graph, i.e., that $p_i \in N_j$ iff $p_j \in N_i$. The neighborhood, G_i^r of a node p_i is all the nodes (including itself) at most r hops away from p_i and $\hat{G}_i^r = G_i^r \setminus \{p_i\}$. Let $g \geq \max_j |G_j^{2r}|$ be a bound, known by the nodes, on the number of nodes within $2r$ hops from any node.

Nodes are driven by a pulse going off every 1 time unit (with respect to its local clock). Pulses are not synchronized between nodes. The pulse frequency,

Constants, and variables:
i : *Constant id of executing processor.*
T, T_{cool}, T_{flood}, κ : *Constants derived from* r, k, λ *and* π. *See Definition 4.*
state ∈ {HEAD, ESCAPING, SLAVE} : *The state of the node.*
timer : *Integer. Timer for escape attempts.*
estart : *Integer. The escape schedule.*
estate ∈ {SLEEP, INIT, FLOOD} : *State for escape attempts.*
heads, *slaves* : *Sets of Id:s tracking what nodes have which role.*
smem, *sendset*, *data* : *Infotuple sets for keeping and forwarding state data.*

External functions and macros:
LBcast(m) : *Broadcasts message* m *to direct neighbors.*
LBrecv(m) : *Receives a message from direct neighbor.*
smallest(a,A) : *Returns the* **min**($|A|$,a) *smallest id:s in* A.
pruneset(A): $maxt \leftarrow \{< j,ji,ttl,ttf> \in A : ttl = \max_\tau \{\tau : < j,ji,\tau,ttf> \in A\}\}$
return $\{< j,ji,ttl,ttf> \in maxt : ttf = \max_\phi \{\phi : < j,ji,ttl,\phi> \in maxt\}\}$
prunemem(A): **return** $\{< j,ji,\xi> \in A : \xi = \max_x \{x : < j,ji,x> \in A\})\}$

Fig. 1. Constants, variables, external functions and macros for the algorithm in Figures 2 and 3

in real time, of a node p_i is denoted ρ_i. For any pair of nodes p_i and p_j the ratio $\rho_i/\rho_j \leq \lambda$, a value is a known to the nodes. Without loss of generality we assume that the frequency of the slowest clock in the system is 1 and thus the clock frequency of any node p_i is in $[1, \lambda]$.

Among π successive messages sent from one node there is at least one message, such that all immediate neighbors $p_j \in N_i$ receive that particular message. Such a message is called a *successful broadcast*. The nodes know the value of π. Apart from that assumption, messages from a node p_i can be lost, be received by a subset of N_i, or received by all nodes in N_i.

3 Self-stabilizing Algorithm for (k, r)-Clustering

The goal of the algorithm is, using as few cluster heads as possible, for each node p_i in the network to have a set of at least k cluster heads within its r-hop neighborhood G_i^r. This is not possible if a node p_i has $|G_i^r| < k$. Therefore, we require that $|C_i^r| \leq k_i$, where $C_i^r \subseteq G_i^r$ is the set of cluster heads in the neighborhood of p_i and $k_i = \min(k, |G_i^r|)$ is the closest number of cluster heads to k that node p_i can achieve. We do not strive for a global minimum. That is too costly. We achieve a local minimum, i.e., a set of cluster heads in which no cluster head can be removed without violating the (k, r) goal.

The basic idea of the algorithm is for cluster heads to constantly broadcast the fact that they are cluster heads and for all nodes to constantly broadcast which nodes they consider to be cluster heads. The set of considered cluster heads consists both of nodes that are known to be cluster heads and, additionally, nodes that are elected to become cluster heads. The content of the broadcasts are forwarded r hops, but in an aggregated form to keep the size of messages down. The election process might establish too many cluster heads. Therefore, there is a mechanism for cluster heads to drop their cluster head roles, to *escape*. Eventually a local minimum of cluster heads forms a total (k,r)-dominating set (or, if not possible given the topology, it fulfills $|C_j^r| \geq k_j$ for any node p_j). The

```
1 on pulse:
2   timer ← (timer + 1)  mod T
3   if estate = SLEEP ∧ ∃t s.t. (i, JOIN, t) ∈ smem then state ← HEAD
4   if state = HEAD then (newheads, newslaves) ← ({i}, ∅)
5   else (newheads, newslaves) ← (∅, {i})
6   for each j ∈ {k |k ≠ i ∧ ∃ ki ≠ JOIN,t s.t. (k,ki,t) ∈ smem} do handlestate(j)
7   (heads, slaves) ← (newheads, newslaves)
8   if state ∈ {HEAD, ESCAPING}                                   /* Escaping */
9     estate ← updateestate()
10    if estate = INIT ∧ state = HEAD ∧ |heads| > k
11      state ← ESCAPING
12      < heads, slaves>  ← < heads \ {i}, slaves ∪ {i}>
13  if state = ESCAPING ∧ estate = SLEEP
14    if ∃t s.t. (i, JOIN, t) ∈ smem then state ← HEAD
15    else state ← SLAVE
16  if state = SLAVE then < estate,estart>  ← < SLEEP,-1>
17  if |heads| < k                                                /* Add heads */
18    heads ← heads ∪ {smallest(k -|heads|, slaves)}
19    slaves ← slaves \ heads
20  for each j ∈ heads                                  /* Join and send state */
21    if j ≠ i then sendset ← pruneset(sendset ∪ {< j, JOIN, r, π> })
22    else state ← HEAD
23  smem ← stepmem(smem)
24  < sendset,data>  ← stepset(pruneset(sendset ∪ {< i, state, r, π> }))
25  LBcast(< i,data> )
26
27 function updateestate:
28  if timer = 0 then estart ← uniformlyrandom({0, 1, . . ., T-T_cool-1})
29  if estart ∈ [0, T-T_cool-1 ]
30    if timer ∈ [estart, estart ] then return INIT
31    else if timer ∈ [estart+1, estart+T_flood-1 ] then return FLOOD
32  return SLEEP
33
34 function handlestate(j):
35  js ← prioritystate(j,smem)
36  if js = HEAD
37    newheads ← newheads ∪ {j}
38    sendset ← pruneset(sendset ∪ {< j, JOIN, r, π> })
39  else if js = ESCAPING ∧ j ∈ heads
40    if |heads| ≤ k
41      newheads ← newheads ∪ {j}
42      sendset ← pruneset(sendset ∪ {< j, VETO, r, π> })
43    else heads ← heads \ {j}
44  newslaves ← (newslaves ∪ {j}) \ newheads
```

Fig. 2. Pseudocode for the self-stabilizing clustering algorithm (1/2)

choice of which nodes that are picked when electing cluster heads is based on node ID:s in order to limit the number of unneeded cluster heads that are elected when new cluster heads are needed.

One could imagine an algorithm that in a first phase adds cluster heads and thereafter in a second phase removes cluster heads that are not needed. To achieve self-stabilization however, we cannot rely on starting in a predefined system state. Recovery from an inconsistent system state might start at any time. Therefore, in our algorithm there are no phases and the mechanism for adding cluster heads runs in parallel with the mechanism for removing cluster heads and none of them ever stops.

At each pulse a node sends out its state (the algorithmic state, i.e., which role it takes in the algorithm) and forwards the states of others. A cluster head node normally has the state HEAD and a non cluster head node always has state

```
46 function prioritystate(j,mem):
47   if ∃ t s.t. (j, HEAD, t) ∈ mem
48     return HEAD
49   if ∃ t s.t. (j, ESCAPING, t) ∈ mem
50     return ESCAPING
51   return SLAVE
52
53 function stepmem(mem):
54   newmem ← ∅
55   for each < j,js,ttk>  in mem
56     ttk ← min(ttk,κ)-1
57     if ttk > 0
58       newmem ← prunemem(
59           newmem ∪ {< j,js,ttk> })
60   return newmem
61
62 function stepset(set):
63   < newset, newdata>  ← < ∅, ∅>
64   for each < j,ji,ttl,ttf>  in set
65     < ttl, ttf>  ← < min(ttl,r), min(ttf,π)-1>
66     if ttf > 0 ∧ ttl > 0 then
67       newset ← pruneset(newset ∪ {< j,ji,ttl,ttf> })
68     if ttf ≥ 0 ∧ ttl > 0 then
69       newdata ← newdata ∪ {< j,ji,ttl> }
70   return < newset, newdata>
71
72 on LBrecv(< k, infoset> ):
73   for each < j,ji,ttl>  ∈ infoset
74     ttl ← min(ttl,r))
75     if ji = VETO
76       if j = i ∧ state = ESCAPING
77         state ← HEAD
78     else if (j ≠ i ∧ ji ≠ JOIN) ∨ (j = i ∧ ji = JOIN)
79       smem ← prunemem(smem ∪ {< j,ji,κ> })
80     if j ≠ i ∧ ttl > 1
81       sendset ← pruneset(sendset ∪ {< j,ji,ttl-1,π> })
```

Fig. 3. Pseudocode for the self-stabilizing clustering algorithm (2/2)

SLAVE. If a node p_i in any pulse finds out that it has less than k cluster heads it selects a set of other nodes that it decides to elect as cluster heads. Node p_i then elects established cluster head nodes and any newly elected nodes by sending a *join* message to them. Any node that is not a cluster head becomes a cluster head if it receives a join addressed to it.

We take a randomized approach for letting nodes try to drop their cluster head responsibility. Time is divided into periods of T pulses. A cluster head node p_i picks uniformly at random one pulse out of the $T - T_{cool}$ first pulses in the period as a possible starting pulse, $estart_i$, for an escape attempt. If p_i has more than k cluster heads in pulse $estart_i$, then it will start an escape attempt. When starting an escape attempt a node sets it state to ESCAPING and keeps it that way for a number of pulses to make sure that all the nodes in G_i^r will eventually know that it tries to escape. A node $p_j \in G_i^r$ that would get fewer than k cluster heads if p_i would stop being a cluster head can veto against the escape attempt. This is done by continuing to regard p_i to be a cluster head and send a VETO back to p_i. If p_j, on the other hand, has more than k cluster heads it would not need to veto. Thus, by accepting the state of p_i as ESCAPING, p_j will not send any join to p_i. After a number of pulses all nodes in $\hat{G}_i^r$ will have had the opportunity to veto the escape attempt. If none of them objected, at that point p_i will get no joins and can set its state to SLAVE.

If an escape attempt by p_i does not overlap in time with another escape attempt it will succeed if and only if $\min_{p_j \in G_i^r} |C_j^r| > k$. If there are overlaps by other escape attempts, the escape attempt by p_i might fail even in cases where $\min_{p_j \in G_i^r} |C_j^r| > k$. The random escape attempt schedule therefore aims to minimize the risk of overlapping attempts.

The pseudocode for the algorithm is described in Figures 2 and 3 with accompanying constants, variables, external functions and macros in Figure 1. At each pulse of a node the lines 1-25 are executed resulting in a message that is broadcast at some time before the next pulse of that node. When a message is being received, the lines 72-81 are executed.

4 Correctness

Due to space constraints, this section contains the theorems and the most important lemmas and an overview on the resulting figures and where they come from. The full set of lemmas and proofs of all lemmas and theorems can be found in [12].

In Section 4.1 we show some basic results that we use further on. In Section 4.2 we will show that within $O(r\pi\lambda^3)$ time we will have $|C_i^r| \geq k_i$ for any node p_i. First we show that this holds while temporarily disregarding the escaping mechanism, and then that it holds for the general case in Theorem 1. In Section 4.3 we will show that a cluster head node p_i can become slave if it is not needed and if it tries to escape undisturbed by other nodes in G_i^{2r}. We continue to show that the set of nodes converges, with high probability, to a local minimum in $O(r\pi\lambda^4 g \log n)$ time in Theorem 2.

Definition 1. *When all system assumptions hold from a point s in time and forward, we say that "we have a legal system execution from s". We denote a pulse of p_i with Γ_x^i for some integer x. Consecutive pulses of p_i have consecutive indices, e.g., Γ_x^i, Γ_{x+1}^i, Γ_{x+2}^i, etc. We denote the time between Γ_x^i and Γ_{x+1}^i with γ_x^i.*

Definition 2. *We define the set of* states *as $\{SLAVE, HEAD, ESCAPING\}$. An* infotuple *is a tuple (j, js, ttx) or (j, js, ttl, ttf), where js is a either a state or one of $\{VETO, JOIN\}$ and is said to be* for node p_j *regardless if p_j is the original sender or final receiver of the infotuple. The ttx field can either be a ttl, the number of hops the info is to be forwarded, or a ttk, the number of pulses for which the infotuple should be kept in smem before being discarded. A ttf field denotes the number of resends that is left to be done for that particular tuple. We say that a state earlier in the list [HEAD, ESCAPING, SLAVE] has priority over a state that is later in that list. We say that an infotuple (j, σ, τ) is memorable$_i$ if and only if either $j \neq i$ and σ is a state, or if $j = i$ and $\sigma = JOIN$ and that it is relevant$_i$ if and only if either it is memorable$_i$ or if $i = j$ and $\sigma = VETO$.*

Definition 3. *A node p_i is said to* handle *a state σ for a node p_j in a pulse Γ_x^i when the handlestate function is called with parameter j at line 6 and the subsequent call to the prioritystate with j as a parameter returns σ, setting $js_i = \sigma$ at line 35.*

4.1 Basic Properties

This section builds up a base on how the algorithm works together with the system settings. First up is the definition of various constants whose value is the result of later lemmas.

Definition 4. *We define* $\kappa = \lceil (2r\pi+1)\lambda \rceil$, $T_{flood} = \lceil r(4\pi+2)\lambda^2 + r(2\pi+2)\lambda \rceil$, $t_s = r(2\pi+1)\lambda^2 + r(\pi+1)\lambda + \lambda - 1$, $t_e = (T_{flood} - 1)\lambda + r(\pi+1)\lambda + \kappa\lambda$, $t_h = \kappa - r(\pi-1)\lambda - 1$, *and* $T_{cool} = \lceil t_e + r(\pi+1)\lambda \rceil$. *Furthermore, we define* $T = T_{es} + T_{cool}$, *where* $T_{es} = \lceil \frac{2g}{\ln 2}(t_s + t_e - 2t_h + 1) \rceil$.

This lemma shows that the algorithm forwards information from any node p_j such that it reaches all nodes in $\hat{G}_j^r$ within time $O(r\pi\lambda)$. The three factors are due to forwarding r hops, only one in π messages are guaranteed to arrive and the clock skew can allows for pulses to be up to λ time apart.

Lemma 1. *Assume that we have a legal system execution from time* $s - r(\pi-1)\lambda$ *and consider a node* p_i *that has a pulse* Γ_x^i *at time* s. *Now, assume that* p_i *has* (k, σ, r, π) *in* $sendset_i$ *just before executing line 24 in* Γ_x^i *and consider a node* $p_j \in G_i^r$, $p_j \neq p_i$ *and a time interval* $\hat{I} = [s - r(\pi-1)\lambda, s + r(\pi+1)\lambda]$. *First, if* (k, σ, τ') *is relevant*$_j$, *there exist a pulse* $\Gamma_y^j \in \hat{I}$ *so that* (k, σ, τ') *is received in* γ_{y-1}^j, *for a* $\tau' \geq 1$. *Second, if* (k, σ, τ') *is memorable*$_j$, *then* $(k, \sigma, \kappa) \in smem_j$ *in* Γ_y^j *just before executing line 2.*

The corollary shows that the mechanisms that keeps data for a certain time, guarantees that eventually nodes in $\hat{G}_j^r$ will see the correct state of node p_j if it stays in that state long enough. Compared to Lemma 1 this introduces another factor $O(\lambda)$ time. This is because if a node wants to make sure that $O(r\pi\lambda)$ time has passed it needs to count $O(r\pi\lambda)$ pulses, but $O(r\pi\lambda)$ pulses can take $O(r\pi\lambda^2)$ time. Building Lemmas on top of each other, this is the mechanism that adds additional factors of λ the further we go in the proof chain.

Corollary 1. *Assume that we have a legal system execution from time* $s - \lambda$ *and consider a node* p_i *that has a pulse* Γ_x^i *at time* s. *Let* $\chi = \lceil r(2\pi+1)\lambda^2 + r(\pi+1)\lambda + 2\lambda \rceil$. *Now assume that a node* p_i, *in each of the pulses* Γ_x^i–$\Gamma_{x+\chi-1}^i$, *adds* (i, σ, π) *to* $sendset_i$ *and does not add* (i, σ', π) *to* $sendset_i$ *for any* $\sigma \neq \sigma'$. *Then, for any node* $p_j \in G_i^r$, p_j *has a pulse* Γ_y^j *at a time* $\hat{s} = s + r(2\pi+1)\lambda^2 + r(\pi+1)\lambda - 1 + t$ *for a* $t \in [0, 2\lambda]$, *in which* p_j *handles* σ *for* p_i. *Furthermore,* $\hat{s}$ *happens between the execution of the pulses* Γ_x^i *and* $\Gamma_{x+\chi-1}^i$.

4.2 Getting Enough Cluster Heads

This section shows that the algorithm will elect enough cluster heads.

Definition 5. *For a node* p_i *to be a* cluster head *is equivalent to* $state_i \in \{HEAD, ESCAPING\}$. *For a node* p_i *to be a slave is equivalent to* $state_i = SLAVE$. *For a node* p_j, *we define* C_j^r *as the set of cluster heads in* G_j^r. *Furthermore, we define* H_x *to be the set of cluster heads in the network at time* x.

Definition 6. *A node* p_i initiates an escape attempt *in a pulse* Γ_x^i *if the condition holds in line 10 and lines 11–12 are executed in* Γ_x^i.

Lemma 2 shows that the escape mechanism works, that a cluster head that is not needed can escape that responsibility.

Lemma 2. *Assume that we have a legal system execution from time* $s - T_{cool}$ *and consider a node* p_i *that initiates an escape attempt in a pulse* Γ_x^i *at time* s.

If all nodes $p_j \in G_i^r$ *have* $|C_j| > k$ *at time* $s + t_h$ *and no node* $p_\ell \in \hat{G}_i^{2r}$, *initiates an escape attempt in any pulse in* $[s - t_e + t_h, s + t_s - t_h]$ *then node* p_i *will set* $state_i$ *to SLAVE in pulse* $\Gamma^i_{x+T_{flood}}$ *and have* $state_i = SLAVE$ *throughout any* $\gamma^i_{x'}$ *or* $\Gamma^i_{x'+1}$ *for any* $x' \geq x + T_{flood}$.

If, on the other hand, there exists a node $p_j \in G_i^r$ *that is having* $|heads_j| \leq k$ *with* $k \in heads_j$ *when* p_j *is first handling ESCAPING for* p_i, *then* p_j *will not set* $state_i$ *to SLAVE in this escape attempt.*

Theorem 1 shows that, within time $T_{cool} + (5r\pi + 4)\lambda \in O(T_{flood}\lambda) = O(r\pi\lambda^3)$ from an arbitrary configuration, all nodes p_i have at least k_i cluster heads within r hops and that the set of cluster heads in the network can only stay the same or shrink from that point on. From Corollary 1 we get the factor $O(\lambda^2)$ time for a node to know that it has reached out. This theorem introduces another factor of $O(\lambda)$ because a node needs to be sure that another node has finished something as discussed previously.

Theorem 1. *Assume that we have a legal system execution from time* s. *Then any node* p_j *will have* k_j *cluster heads from time* $s+T_{cool}+(3r\pi+2)\lambda$ *and onward. Moreover, a node that is not in* H_t *for a time* $t \geq s+T_{cool}+(5r\pi+4)\lambda$ *can not be in* $H_{t'}$ *for a* $t' \geq t$ *and consequently* $|H_{t'}| \leq |H_t|$ *for any* $s+T_{cool}+(5r\pi+4)\lambda \leq t \leq t'$.

4.3 Convergence to a Local Minimum

Lemma 2 shows that a cluster head node that is not needed can escape the cluster head responsibility if it does not interfere with escape attempts by other nodes. This section shows that the set of cluster heads converges to a local minimum. We first show that an unneeded cluster head node can escape, with high probability (Lemma 3) in $O(T\lambda) = O(gr\pi\lambda^4)$ time. The extra λ is due to the usual reason and $T \in O(gr\pi\lambda^3)$. The factor g is a bound on the number of nodes that could interfere with a given escape attempt. It is part of T to give a node a constant probability of escaping in its one try in a period of T time (given that it is not needed as a cluster head).

Lemma 3. *Assume that we have a legal system execution from time* s *and consider a node* p_i *that is a cluster head. Assume that* $|C_j^r| > k$ *holds for all nodes* $p_j \in \hat{G}_i^r$ *from time* $s + T_{cool} + (5r\pi + 4)\lambda$ *and as long as* p_i *remains a cluster head. Then, node* p_i *will have* $state_i = SLAVE$ *after time* $s + T_{cool} + (5r\pi + 4)\lambda + (\beta + 1)T\lambda$ *with at least probability* $1 - 2^{-\beta}$.

Theorem 2, shows that with high probability the entire network reaches a local minimum within $O(r\pi\lambda^4 g \log n)$ time.

From Theorem 1 we got that all nodes p_i have at least k_i cluster heads within r hops in $T_{cool} + (3r\pi + 2)\lambda$ time after an arbitrary configuration.

Theorem 2 shows that with at least probability $1 - 2^{-\alpha}$ the set of cluster heads in the network stabilizes to a local minimum within $s + T_{cool} + (5r\pi + 4)\lambda + (\alpha + \log n + 1)T\lambda$ time. The factor $O(\log n)$ is multiplied by the result from Lemma 3 because we go from probabilistic guarantee that one specific node gets an uninterrupted escape attempt to that all cluster heads get such an attempt. And the number of cluster heads is bounded by the number, n, of nodes in the network.

Theorem 2. *Assume that we have a legal system execution from time s. With at least probability $1-2^{-\alpha}$, by time $\hat{s} = s+T_{cool}+(5r\pi+4)\lambda+(\alpha+\log n+1)T\lambda$ there will be no cluster head node p_i in the network for which $\min_{p_j \in G_i}(|C_j^r|) > k$ holds, and $H_{\hat{s}+t} = H_{\hat{s}}$ holds for any positive t.*

5 Discussion

To experimentally test performance, we did simulations of the algorithm for various settings of k and r. We placed 40 nodes with a communication radius of 1 uniformly at random in a 5 by 5 rectangular area. From our experiments we concluded that using a g that gives us 95% guarantees of being an upper bound on every $|G_j^{2r}|$ for any given p_j, is not required to get good performance. The calculated bounds are not tight. In the experiments we have therefore used a tenth of that value for g instead.

In addition, we performed experiments on recovery from small changes to the topology from a converged system state. The convergence times from a newly started network ("Start") is compared in Figure 4 with the convergence times after a change to a initially converged network. We investigate 10% added nodes ("Add"), 10% removed nodes ("Remove") or 10% moved nodes ("Move").

We can see that the least obtrusive change to the topology is removed nodes. The chance is good that a removed node is not a cluster head and thus do not upset the balance. An add is more expensive than a remove. Nodes might end up in an area where there is not so many cluster heads and therefore have to start elect new nodes. A move is like both a remove and an add. Therefore, it is anticipated that this case converges slower than the ones with only adds or only removes.

The flooding of messages makes sure that if there exist multiple paths of at most length r between a node p_i and a node p_j then joins and state updates will traverse all possible paths. This can give us higher fault tolerance if there are communication disturbances on some links (i.e., between some immediate neighbors) and also higher availability for nodes to reach their cluster heads.

The multiple paths can also give applications higher security if some nodes in the network can be compromised. If there is at least one path of at most r hops between a node p_i and a node p_j that is not passing through any compromised nodes then the flooding makes sure that node p_i and p_j gets to know about each other. Moreover, if p_j wants p_i to be cluster head then the compromised nodes

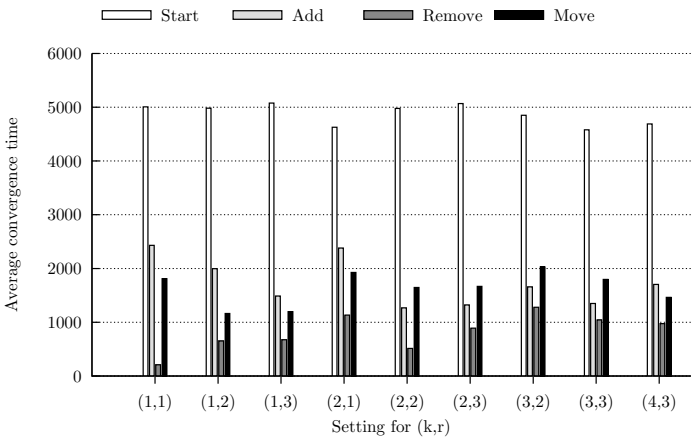


Fig. 4. Convergence times from a fresh start, after 10% node additions, after 10% node removes and after 10% node moves

cannot stop that. If nodes add information to the messages about the paths they have taken during message forwarding then the nodes get to know about the multiple paths. With this knowledge they can in an application layer use as diverse paths as possible to communicate with their cluster heads. Thus even if a compromised node is on the path to one cluster head and drops messages or do other malicious behavior there can be other cluster heads for where there is no compromised nodes on the chosen paths.

Consider a compromised node p_c that can lie and not follow protocol. First assume that p_c cannot introduce node id:s that does not exist (Sybil attacks, [13]) or node id:s for nodes that are not within G_c^r (wormhole attacks, [8]) and that p_c cannot do denial of service attacks. Then p_c can make any or all nodes within G_c^r become and stay cluster heads by sending joins to them or having them repeatedly go on and off cluster head duty over time by alternating between sending joins and letting the node escape. Consider a node p_i that is a cluster head and has a path to a node p_j of length $\leq r$ hops that does not pass through p_c. In this situation p_c can not give the false impression that p_i is not a cluster head as HEAD takes precedence over ESCAPING that takes precedence over SLAVE at message receipt. If p_c on the other hand is in a bottleneck between nodes without any other paths between them then it can lie about a node p_ℓ being a cluster heads and refuse to forward any joins to p_ℓ. Now if we assume that p_c is not restricted in what id:s it can include in false messages it can convince a node p_ℓ that nodes not in G_ℓ^r are cluster heads. In the worst case it can eventually make p_ℓ rely exclusively on non-existent cluster heads with paths that all go through p_c. In any case the influence by a compromised node p_c is contained within G_c^{2r} as the maximum *ttl* of a message is r and is enforced at message receipt.

6 Conclusions

We have presented a self-stabilizing (k, r)-clustering algorithm for ad-hoc networks that can deal with a bounded amount of message loss, and that merely requires a bound on the rate differences between pulses of different nodes in the network. The algorithm makes sure that, within $O(r\pi\lambda^3)$ time, all nodes have at least k cluster heads (when possible) and it stabilizes within $O(r\pi\lambda^4 g \log n)$ time with high probability. We have also discussed how the algorithm can help us with fault tolerance and security.

References

1. Abbasi, A.A., Younis, M.: A survey on clustering algorithms for wireless sensor networks. Comput. Commun. 30(14-15), 2826–2841 (2007)
2. Afek, Y., Brown, G.: Self-stabilization over unreliable communication media. Distributed Computing 7, 27–34 (1993)
3. Caron, E., Datta, A.K., Depardon, B., Larmore, L.L.: A Self-stabilizing K-Clustering Algorithm Using an Arbitrary Metric. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 602–614. Springer, Heidelberg (2009)
4. Chen, Y.P., Liestman, A.L., Liu, J.: Clustering Algorithms for Ad Hoc Wireless Networks, ch. 7, vol. 2, pp. 154–164. Nova Science Publishers (2004)
5. Dolev, S.: Self-Stabilization. MIT Press (2000)
6. Dolev, S., Tzachar, N.: Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. Theor. Comput. Sci. 410(6-7), 514–532 (2009)
7. Fu, Y., Wang, X., Li, S.: Construction k-dominating set with multiple relaying technique in wireless mobile ad hoc networks. In: CMC 2009, pp. 42–46. IEEE Computer Society, Washington, DC (2009)
8. Hu, Y., Perrig, A., Johnson, D.B.: Wormhole detection in wireless ad hoc networks. Technical report, Rice University, Department of Computer Science (2002)
9. Johnen, C., Nguyen, L.H.: Robust self-stabilizing weight-based clustering algorithm. Theor. Comput. Sci. 410(6-7), 581–594 (2009)
10. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
11. Larsson, A., Tsigas, P.: A self-stabilizing (k,r)-clustering algorithm with multiple paths for wireless ad-hoc networks. In: ICDCS 2011, Minneapolis, MN, USA (2011)
12. Larsson, A., Tsigas, P.: Self-stabilizing (k,r)-clustering in clock rate-limited systems. TR 2012:05, Computer Science and Engineering, Chalmers University of technology (2012)
13. Newsome, J., Shi, E., Song, D., Perrig, A.: The sybil attack in sensor networks: analysis & defenses. In: IPSN 2004, pp. 259–268. ACM, New York (2004)
14. Ravelomanana, V.: Distributed k-Clustering Algorithms for Random Wireless Multihop Networks. In: Lorenz, P., Dini, P. (eds.) ICN 2005, Part I. LNCS, vol. 3420, pp. 109–116. Springer, Heidelberg (2005)
15. Spohn, M.A., Garcia-Luna-Aceves, J.J.: Bounded-distance multi-clusterhead formation in wireless ad hoc networks. Ad Hoc Netw. 5(4), 504–530 (2007)
16. Sun, K., Peng, P., Ning, P., Wang, C.: Secure distributed cluster formation in wireless sensor networks. In: ACSAC 2006, pp. 131–140. IEEE Computer Society, Washington, DC (2006)
17. Wu, Y., Li, Y.: Construction algorithms for k-connected m-dominating sets in wireless sensor networks. In: MobiHoc 2008, pp. 83–90. ACM, New York (2008)

Increasing the Power of the Iterated Immediate Snapshot Model with Failure Detectors

Michel Raynal[1,2] and Julien Stainer[1]

[1] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
[2] Institut Universitaire de France

Abstract. The base distributed asynchronous read/write computation model is made up of n asynchronous processes which communicate by reading and writing atomic registers only. The distributed asynchronous iterated model is a more constrained model in which the processes execute an infinite number of rounds and communicate at each round with a new object called immediate snapshot object. Moreover, in both models up to $n-1$ processes may crash in an unexpected way. When considering computability issues, two main results are associated with the previous models. The first states that they are computationally equivalent for decision tasks. The second states that they are no longer equivalent when both are enriched with the same failure detector.

This paper shows how to capture failure detectors in each model so that both models become computationally equivalent. To that end it introduces the notion of a "strongly correct" process which appears particularly well-suited to the iterated model, and presents simulations that prove the computational equivalence when both models are enriched with the same failure detector. The paper extends also these simulations to the case where the wait-freedom requirement is replaced by the notion of t-resilience.

1 Introduction

Base read/write model and tasks. The base asynchronous read/write (ARW) computation model consists of n asynchronous sequential processes that communicate only by reading and writing atomic registers. Moreover, any number of processes (but one) are allowed to crash in an unexpected way.

A decision task is the distributed analogous of the notion of a function encountered in sequential computing. Each process starts with its own input value (without knowing the input values of the other processes). The association of an input value with each process define an input vector of the task. Each process has to compute its own output value in such a way that the vector of output values satisfies a predefined input/output relation (this is the relation that defines the task). The most famous distributed task is the consensus task: each process proposes a value and processes have to decide the very same value which has to be one of the proposed values. The progress condition that is usually considered is called *wait-freedom* [4]. It requires that any process that does not crash eventually decides a value. It has been shown that the consensus task cannot be wait-free solved in the ARW model. The tasks that can be wait-free solved in this base model are called *trivial* tasks.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 231–242, 2012.

The iterated immediate snapshot (IIS) model and its power. The fact that, in the ARW model, a process can issue a read or write on any atomic register at any time makes difficult to analyze the set of runs that can be generated by the execution of an algorithm that solves a task in this model.

To make such analyses simpler and obtain a deeper understanding of the nature of asynchronous runs, Borowsky and Gafni have introduced the *iterated immediate snapshot* (IIS) model [2]. In this model, each process (until it possibly crashes) executes asynchronously an infinite number of rounds and, in each round, processes communicate through a one-shot *immediate snapshot* object [1] associated with this round. Such an object provides the processes with a single operation denoted write_snapshot() that a process can use only once. This operation allows the invoking process to deposit a value in the corresponding object and obtains a snapshot of the values deposited into it.

A *colorless* decision task is a task such that any value decided by a process can be decided by any number of processes. The main result associated with the IIS model is the following one: A colorless decision task can be wait-free solved in the ARW model if and only if it can be wait-free solved in the IIS model (Borowsky and Gafni [2]).

Enriching a model with a failure detector. One way to enrich the base read/write model in order to obtain a stronger model consists in providing the processes with operations whose computational power in presence of asynchrony and process crashes is stronger than the one of the base read or write operations [4]. Another way to enrich the base read/write model consists in adding to it a failure detector [3].

A failure detector is a device that provides each process with a read-only variable that gives it information on failures. According to the type and the quality of this information, several classes of failure detectors can be defined. As an example, a failure detector of the class Ω provides each process p_i with a read-only local variable denoted $\texttt{leader}_i$ that contains always a process identity. The property associated with these read-only local variables is the following: there is an unknown but finite time after which all the variables $\texttt{leader}_i$ contain forever the same identity and this identity is the one of a non-faulty process. A failure detector is non-trivial if it cannot be built in the base read/write model (i.e., if it enriches the system with additional power).

A natural question is then the following: Are the ARW model and the IIS model still equivalent for wait-free task solvability when they are enriched with the same non-trivial failure detector? It has been shown by Rajsbaum, Raynal and Travers that the answer to this question is "no" [6]. It follows that, from a computability point of view, the ARW model enriched with a non-trivial failure detector is more powerful than the IIS model enriched with the same failure detector.

An approach aiming at introducing the power of failure detectors into the IIS model has been investigated in [5]. This approach consists in requiring some property P to be satisfied by the successive invocations of write_snapshot() issued on the sequence of immediate snapshot objects. Hence the name *iterated restricted immediate snapshot* (IRIS) given to this model. For each failure detector class taken separately, this approach requires (a) to associate a specific property P with the considered failure detector class, (b) to design an ad hoc simulation of the write_snapshot() operation suited to this failure detector class in order to simulate IIS in ARW and (c) design a specific simulation of the output of the failure detector to to simulate ARW in IIS.

Content of the paper. Let C be a failure detector class defined in the context of the base ARW model. This paper is motivated by the following question: Is it possible to associate with C (in a systematic way) a failure detector class C^* such that the ARW model enriched with C and the IIS model enriched with C^* are equivalent for wait-free task solvability? The contributions of the paper are the following:

- The answer to the previous question is based on a simple modification of the definition of what is a *correct* process (i.e., a process that does not crash in a run of the base read/write model). The notion of a correct process is replaced in the IIS model by what we call a *strongly correct* process. Such a process is a process that does not crash and all of whose invocations of write_snapshot() are seen (directly or indirectly) by all other non-crashed processes. (Although it is not explicitly defined in [5], the notion of a *strongly correct* process is implicitly used in the proof of a theorem in that paper.)
  Given this definition, and a failure detector class C designed for the ARW model, its IIS counterpart C^* is obtained by a simple and systematic replacement of the words "correct process(es)" by "strongly correct process(es)" in the definition of C.
- An immediate benefit of the previous definition is the fact that, when we want to simulate ARW in IIS, we can directly benefit from Borowsky and Gafni's simulation of the read and write operations defined in [2]. (The only addition that has to be done concerns the local outputs of the corresponding failure detector C.)
- Given the ARW model enriched with a failure detector class C, the paper presents a generic simulation of the IIS model enriched with C^*. This simulation is generic in the sense that it works for a large family of failure detectors. The simulation algorithm has only to be instantiated with a predicate associated with the corresponding failure detector C.
- An interesting consequence of the fact that, given a failure detector class C, we have "for free" a corresponding failure detector for the IIS model, makes very simple the understanding of IIS enriched with a failure detector.
- The paper also generalizes the previous wait-free simulations to t-resilient simulations (wait-freedom is $(n-1)$-resilience).

Due to page limitation this paper is an extended abstract. More developments and proofs will be found in [7].

2 Base Definitions

Process model. The n asynchronous sequential processes are denoted p_1, ..., p_n. Any number of processes may crash. A process that crashes in a run is *faulty* in that run, otherwise it is *correct*. A correct process executes an infinite number of steps. Given an execution, let $\mathcal{C}$ denote the set of processes that are correct in that execution and $\mathcal{F}$ the set of the faulty ones.

Decision tasks. A *decision task* is a one-shot decision problem specified in terms of an input/output relation Δ. Each process starts with a private input value and must eventually decide on a private output value. An input vector $I[1..n]$ specifies the input

$I[i] = v_i$ of each process p_i. Similarly, an output vector $O[1..n]$ specifies a decision value $O[j]$ for each process p_j. A task is defined by a set of input vectors and a relation Δ that describes which output vectors are correct for each input vector I.

Base read/write model. This model, denoted $\mathcal{ARW}_n[\emptyset]$ in the following, has been presented in the introduction. Instead of atomic read/write registers, we consider here that the processes communicate through *snapshot* objects.This is at no additional computability cost as the operations on a snapshot object can be wait-free implemented from single-writer/multi-reader atomic read/write registers. Given a snapshot object S, these operations are denoted S.write() and S.snapshot(). S is initially empty. When p_i invokes S.write(v), it adds the pair $\langle i, v\rangle$ to S and suppresses the previous pair $\langle i, -\rangle$ if any. When it invokes S.snapshot(), p_i obtains a set containing all the pairs $\langle k, v_k\rangle$ currently present in S. Such a set is called a *view* that we denote arw_view_i. A snapshot object S is *atomic* (we also say said that it is *linearizable*).

One-shot immediate snapshot object. Such an object is similar to a snapshot object where the write() and S.snapshot() operations are encapsulated into a single operation denoted write_snapshot() (where the write is executed just before the snapshot). Moreover, *one-shot* means that, given an object, each process invokes write_snapshot() at most once. The invocations of IS.write_snapshot() are not linearizable but *set-linearizable*. Let us consider a process p_i that invokes IS.write_snapshot(v_i) where v_i is the value it wants to write into IS. When it returns from its invocation, p_i obtains a *view* of the object that we denote iis_view_i, Moreover, let us define $iis\_view_i = \emptyset$ if p_i never invokes IS.write_snapshot(). A one-shot immediate snapshot object is defined by the following properties associated with the views obtained by the processes.

- Self-inclusion. $\forall\, i :\ \langle i, v_i\rangle \in iis\_view_i$.
- Containment. $\forall\, i, j :\ (iis\_view_i \subseteq iis\_view_j)\ \vee\ (iis\_view_j \subseteq iis\_view_i)$.
- Immediacy. $\forall\, i, j :\ (\langle i, v_i\rangle \in iis\_view_j) \Rightarrow\ (iis\_view_i \subseteq iis\_view_j)$.

The first property states that a process sees its write. The second property states that the views are totally ordered by containment. The third property states that, when a process invokes IS.write_snapshot(), its snapshot is scheduled just after its write. The operation write_snapshot() can be wait-free implemented in $\mathcal{ARW}_n[\emptyset]$.

The iterated immediate snapshot model. In the base IIS model (denoted $\mathcal{IIS}_n[\emptyset]$), the shared memory is made up of an infinite number of immediate snapshot objects $IS[1]$, $IS[2]$, ... These objects are accessed sequentially and asynchronously by the processes according to the following round-based pattern executed by each process p_i. The variable r_i is local to p_i and denotes its the current round number.

$r_i \leftarrow 0$; $\ell s_i \leftarrow$ initial local state of p_i (including its input, if any);
repeat forever (asynchronous IIS rounds)
  $r_i \leftarrow r_i + 1$;
  $iis\_view_i \leftarrow IS[r_i]$.write_snapshot($\ell s_i$);
  computation of a new local state ℓs_i (which contains iis_view_i)
end repeat.

Full information algorithm. As the aim of the IIS model is to address computability issues (and not efficiency), we consider *full information* algorithms for this computation model. This means that, at each round r, a process p_i writes its current local state ℓs_i into $IS[r]$. Consequently, the view it obtains from $IS[r].\mathsf{write\_snapshot}(\ell s_i)$ contains all its causal past (hence the name *full information*).

Failure detectors. A failure detector is a device that provides processes with information on failures [3]. As already said in the introduction, several classes of failure detectors can be defined according to the type and the quality of the information given to the processes. We consider here that the information given to each process is a set of process indexes. This information is given by a failure detector to a process through a local read-only variable.

Classes of failure detectors that eventually output only correct processes. We consider three classes of such failure detectors: the classes P of perfect failure detectors and $\diamond P$ of eventually perfect failure detectors [3] and the class Σ of quorum failure detectors.

A failure detector of the class P provides each process p_i with a set $\mathtt{trusted}_i$ that, at any time τ, contains all the processes that have not crashed by time τ and eventually contains only correct processes. (The traditional definition of P provides each process p_i with a set $\mathtt{faulty}_i$ that does not contain a process before it crashes and eventually contains all faulty processes. It is easy to see that $\mathtt{trusted}_i = \{1 \ldots, n\} \setminus \mathtt{faulty}_i$.) The class $\diamond P$ is weaker than the class P. Namely, there is an arbitrary long finite period during which the sets $\mathtt{trusted}_i$ can contain arbitrary values and when this period terminates a failure detector of $\diamond P$ behaves as a failure detector of P.

A failure detector of the class Σ provides each process p_i with a set $\mathtt{qr}_i$ that eventually contains only correct processes and is such that the value of $\mathtt{qr}_i$ at any time τ and the value of any $\mathtt{qr}_j$ at any time τ' have a non-empty intersection.

Classes of failure detectors that eventually output correct and possibly faulty processes. We consider here the class of eventual leaders failure detectors denoted Ω_k. This class class is a straightforward generalization of the failure detector class Ω. Actually, Ω_1 is Ω which has been shown to be the weakest failure detector class for solving the consensus task in shared memory systems.

A failure detector of the class Ω_k provides each process p_i with a read-only local variable $\mathtt{leaders}_i$ that always contains k process indexes. Moreover, the local variables $\mathtt{leaders}_i$ are such that, after some unknown but finite time τ, they all contain forever the same set of k process indexes and at at least one of these indexes is the one of a correct process. Let us notice that, before time τ, the sets $\mathtt{leaders}_i$ can contain arbitrarily changing sets of k process indexes.

3 Strongly Correct Processes (wrt the IIS Model)

Motivation. When considering the base read/write model, if a process issues a write into a snapshot object S, the value it has written can be read by any process that invokes $S.\mathsf{snapshot}()$. This is no longer the case in the IIS model. This observation motivates

the definition of a *strongly correct* process. Such a process is a process whose writes into the immediate snapshot objects are seen (directly or indirectly) infinitely often by the all correct processes. A process that is not strongly correct is consequently a process such that only a finite number of its writes into immediate snapshot objects are eventually propagated to all the correct processes.

Formal definition. Let $iis\_view_j[r]$ be the view obtained by p_j at round r. Let $\mathcal{SC}_0$ be the set defined as follows ($\mathcal{C}$ denotes the set of correct processes):

$$\mathcal{SC}_0 \stackrel{def}{=} \{\ i \text{ such that } |\{r \text{ such that } \forall\ j \in \mathcal{C} :\ \exists\ \langle i, -\rangle \in iis\_view_j[r]\}| = \infty\ \},$$

i.e., $\mathcal{SC}_0$ is the set of processes that have issued an infinite sequence of (not necessarily consecutive) invocations of write_snapshot() and these invocations have been seen by each correct process (this is because these invocations are set-linearized in the first position when considering the corresponding one-shot immediate snapshot objects).

Let us observe that, as soon as we assume that there is at least one correct process, it follows from the fact that the number of processes is bounded that $|\mathcal{SC}_0| \neq 0$. Given $k > 0$ let us recursively define $\mathcal{SC}_k$ as follows:

$$\mathcal{SC}_k \stackrel{def}{=} \{\ i \text{ such that } |\{r \text{ such that } \exists\ j \in \mathcal{SC}_{k-1} :\ \exists\ \langle i, -\rangle \in iis\_view_j[r]\}| = \infty\ \}.$$

Hence, $\mathcal{SC}_k$ contains all the correct processes that have issued an infinite sequence of (not necessarily consecutive) invocations of write_snapshot() which have been seen by at least one process of $\mathcal{SC}_{k-1}$. It follows from the self-inclusion property of the views and the definition of $\mathcal{SC}_k$ that $\mathcal{SC}_0 \subseteq \mathcal{SC}_1 \subseteq \cdots$. Moreover, as all the sets are included in $\{1, \ldots, n\}$, there is some K such that $\mathcal{SC}_0 \subseteq \mathcal{SC}_1 \subseteq \cdots \subseteq \mathcal{SC}_K = \mathcal{SC}_{K+1} = \mathcal{SC}_{K+2} = \cdots$.

$\mathcal{SC}_K$ defines the set of of strongly correct processes which is denoted $\mathcal{SC}$. This is the set of processes that have issued an infinite sequence of (not necessarily consecutive) invocations of write_snapshot() which have been propagated to all the correct processes.

IIS enriched with a failure detector. Let C be a failure detector class. C^* denotes the same failure detector class where the word "correct" is replaced by the word "strongly correct". Moreover, $\mathcal{IIS}_n[C^*]$ denotes the IIS model enriched with a failure detector of the class C^* where, during each round r, a process p_i reads its failure detector variable at the beginning of round r and saves its value fd_i in its local state ℓs_i before writing it into $IS[r]$.

4 From $\mathcal{IIS}_n[C^*]$ to $\mathcal{ARW}_n[C]$

This section describes a simulation in $\mathcal{IIS}_n[C^*]$ of a run of an algorithm designed for $\mathcal{ARW}_n[C]$. Except for the simulation of the detector output, this simulation is from [2]. In order not to confuse a simulated process in $\mathcal{ARW}_n[C]$ and its simulator in $\mathcal{IIS}_n[C^*]$, the first one is denoted p_i while the second one is denoted q_i.

4.1 Description of the Simulation

It is assumed, without loss of generality, that the simulated processes communicate through a single snapshot object S. A simulator q_i is associated with each simulated process p_i. It locally executes the code of p_i and uses the algorithms described in Figure 1 when p_i invokes $S.\mathsf{write}(-)$, $S.\mathsf{snapshot}()$ or queries the failure detector.

Immediate snapshot objects of $\mathcal{IIS}_n[C^*]$. These objects are denoted $IS[1], IS[2], ...$ Each object $IS[r]$ stores a set of triples (this set is denoted ops_i in Figure 1). If the triple $(j, sn, x) \in ops_i$, then the simulator q_i knows that the process p_j (simulated by q_j) has issued its sn-th invocation of an operation on the simulated snapshot object S; $x \neq \perp$ means that this invocation is $S.\mathsf{write}(x)$ while $x = \perp$ means that it is $S.\mathsf{snapshot}()$.

Local variables of a simulator q_i. The variable r_i contains the current round number of the simulator q_i. It is increased before each invocation of $IS_n[r_i].\mathsf{write\_snapshot}(ops_i)$ (line 3). As this is the only place where, during a round, a simulator invokes the operation $\mathsf{write\_snapshot}()$, the simulators obey the IIS model.

The local variable sn_i is a sequence number that measures the progress of the simulation by q_i of the process p_i. It is increased at line 1 when p_i starts simulating a new invocation of $S.\mathsf{write}()$ or $S.\mathsf{snapshot}()$ on behalf on p_i.

As already indicated, the local variable ops_i contains the triples associated with all the invocations of $S.\mathsf{write}()$ and $S.\mathsf{snapshot}()$ that have been issued by the processes and are currently known by the simulator q_i. This local variable (which can only grow) is updated at line 1 when q_i starts simulating the next operation $S.\mathsf{write}()$ or $S.\mathsf{snapshot}()$ issued by p_i or at line 4 when q_i learns operations on the snapshot object S issued by other processes.

The local variable iis_view_i stores the value returned by the last invocation of $IS[r_i].\mathsf{write\_snapshot}()$ issued by the simulator q_i (line 3). When simulating an invocation of $S.\mathsf{snapshot}()$ issued by p_i, q_i computes for each simulated process p_j the sequence number max_snj_i (line 11) of the last value it knows (saved in vj_i at line 12) that has been written by p_j in the snapshot object S. This allows q_i to compute the view arw_view_i (line 13) that it returns (line 16) as the result of the invocation of $S.\mathsf{snapshot}()$ issued by p_i.

The local variable fd_i is used to store the last value obtained by the simulator q_i from its read-only local failure detector variable denoted $C^*.\mathsf{read}()$.

Simulation of $S.\mathsf{write}(v)$. To simulate the invocation of $S.\mathsf{write}(v)$ issued by p_i, the simulator q_i invokes the internal operation $\mathsf{publicize\&progress}(v)$. It first increments sn_i and adds the triple (i, sn_i, v) to ops_i (line 1). Then, the simulator repeatedly invokes $\mathsf{write\_snapshot}(ops_i)$ on successive immediate snapshot objects and enriches its set of triples ops_i (lines 2-4) until it obtains a view iis_view_i in which all the simulators it sees are aware of the invocation of the operation $S.\mathsf{write}(v)$ that it is simulating (line 6).

Simulation of $S.\mathsf{snapshot}()$. To simulate an invocation of $S.\mathsf{snapshot}()$ issued by p_i, the simulator q_i first invokes $\mathsf{publicize\&progress}(\perp)$. When this invocation terminates, q_i knows that all the simulators it sees in the last view iis_view_i it has obtained are

aware of its invocation of S.snapshot(). Moreover, as we have seen, the execution of publicize&progress($\perp$) makes q_i aware of operations simulated by other simulators.

Then the simulator q_i browses all the operations it is aware of in order to extract, for each simulated process p_j, the last value effectively written by p_j (lines 9-15). This (non-$\perp$) value is extracted from the triple with the largest sequence number among all those that appear in all the sets ops_k that belong to the view iis_view_i returned to q_i by its last invocation of write_snapshot().

Init: $ops_i \leftarrow \emptyset$; $r_i \leftarrow 0$; $sn_i \leftarrow 0$; $iis\_view_i \leftarrow \emptyset$; $fd_i \leftarrow C^*$.read().

internal operation publicize&progress (x) **is**
(1) $sn_i \leftarrow sn_i + 1$; $ops_i \leftarrow ops_i \cup \{(i, sn_i, x)\}$;
(2) **repeat** $r_i \leftarrow r_i + 1$; $fd_i \leftarrow C^*$.read();
(3) $iis\_view_i \leftarrow IS[r_i]$.write_snapshot$(ops_i)$;
(4) $ops_i \leftarrow \bigcup_{\langle k, ops_k\rangle \in iis\_view_i} ops_k$
(5) **until** $\big((i, sn_i, x) \in \bigcap_{\langle k, ops_k\rangle \in iis\_view_i} ops_k\big)$ **end repeat**;
(6) return().

operation S.write(v) **is** publicize&progress (v); return().

operation S.snapshot() **is**
(7) publicize&progress $(\perp)$;
(8) $arw\_view_i \leftarrow \emptyset$;
(9) **for each** j **in** $\{1, \ldots, n\}$ **do**
(10) **if** $\big(\exists v \mid (j, -, v) \bigcap_{\langle k, ops_k\rangle \in iis\_view_i} ops_k \wedge v \neq \perp\big)$
(11) **then** $max\_snj_i \leftarrow \max\{sn \mid (j, sn, v) \in \bigcap_{\langle k, ops_k\rangle \in iis\_view_i} ops_k \wedge v \neq \perp\}$;
(12) $vj_i \leftarrow v$ such that $(j, max\_snj_i, v) \in ops_i$;
(13) $arw\_view_i \leftarrow arw\_view_i \cup \{\langle j, vj_i\rangle\}$
(14) **end if**
(15) **end for**;
(16) return (arw_view_i).

operation C.read() **is** return (fd_i).

Fig. 1. Simulation of $\mathcal{ARW}_n[C]$ in $\mathcal{IIS}_n[C^*]$: code for a simulator q_i (extended from [2])

Simulation of C.read(). When a process p_i reads its local failure detector output, the simulator q_i simply returns it the current value of fd_i.

4.2 From Strongly Correct Simulators to Correct Simulated Processes

Strongly correct vs weakly correct simulators. Let $\mathcal{WC} = \mathcal{C} \setminus \mathcal{SC}$ (the set of weakly correct simulators). It follows from the definition of the strongly correct simulators that, for any simulated process p_i whose simulator q_i is such that $i \in \mathcal{WC}$, there is a round $rmin_i$ such that, $\forall j \in \mathcal{SC}, \forall r \geq rmin_i : \langle i, -\rangle \notin iis\_view_j[r]$, which means that, for $r \geq rmin_i$, no invocation $IS[r]$.write_snapshot() issued by the simulator q_i is seen by a strongly correct simulator.

This means that, after $rmax = \max\{rmin_i\}_{i \in \mathcal{WC}}$ and after all simulator crashes have occurred, the invocations of write_snapshot() by the simulators of $\mathcal{SC}$ are always set-linearized strictly before the ones of the simulators of $\mathcal{WC}$. Said differently, there is a round after which no strongly correct simulator ever receives information from a weakly correct simulator. From the point of view of a strongly correct simulator, any weakly correct simulator appears as a crashed simulator. Differently, any weakly correct simulator receives forever information from all the strongly correct simulators.

Crashed and slow IIS simulators simulate crashed ARW processes. An important feature of the simulation described in Figure 1 is that, not only the crash of a simulator q_i gives rise to the crash of the associated simulated process p_i, but a slow simulator q_j entails the crash of its simulated process p_j.

To summarize. When simulating $\mathcal{ARW}_n[C]$ on top of $\mathcal{IIS}_n[C^*]$, we have the following: (a) a faulty or weakly correct simulator q_i gives rise to a faulty simulated process p_i and (b) a strongly correct process gives rise to a correct simulated process p_i in $\mathcal{ARW}_n[C]$. The next theorem captures the previous discussion.

Theorem 1. *Let A be an algorithm solving a colorless task in the $\mathcal{ARW}_n[C]$ model. Let us consider an execution of A simulated in the $\mathcal{IIS}_n[C^*]$ model by the algorithms S.write(), S.snapshot() and C.read() described in Figure 1. A process p_i is correct in the simulated execution iff its simulator q_i is strongly correct in the simulation.*

5 From $\mathcal{ARW}_n[C]$ to $\mathcal{IIS}_n[C^*]$

This section presents a generic simulation of $\mathcal{IIS}_n[C^*]$ in $\mathcal{ARW}_n[C]$. Its generic dimension lies in the fact that C can be any failure detector class cited in Section 2 (namely, P, Σ, $\diamond P$, Ω, Ω_k and others such as S, $\diamond S$ [3] and $\diamond S_x$). As far terminology is concerned, q_i is used to denote a simulated IIS process while p_i is used to denote the corresponding ARW simulator process. The simulation is described in Figure 2. Differently from the simulation described in Figure 1, the algorithms of Figure 2 are not required to be full-information algorithms.

5.1 Description of the Simulation

The simulated model $IS[1]$, $IS[2]$, ... denote the infinite sequence of one-shot immediate snapshot objects of the simulated IIS model. Hence, a simulated process q_i invokes $IS[r]$.write_snapshot() and C^*.read().

Shared objects of the simulation The simulation uses an infinite sequence of objects $S[1]$, $S[2]$, ... The object $S[r]$ is used to implement the corresponding one-shot immediate snapshot object $IS[r]$. Each object $S[r]$ can be accessed by two operations, which are denoted collect() and arw_write_snapshot(). The later is nothing else than the operation write_snapshot() (which satisfies the self-inclusion, containment and immediacy properties defined in Section 2). It is prefixed by "arw" in order not to be confused with the operation of the IIS model that it helps simulate. The operation collect() is similar

to the operation snapshot(), except that it is not required to be atomic. It consists in an asynchronous scan of the corresponding $S[r]$ object which returns the set of pairs it has seen in $S[r]$. Both collect() and arw_write_snapshot() can be wait-free implemented in $\mathcal{ARW}_n[\emptyset]$.

FD_VAL is an array of single-writer/multi-reader atomic registers. The simulator p_i stores in the register $FD\_VAL[i]$ the last value it has read from its local failure detector variable which is denoted C.read().

```
operation IS[r].write_snapshot(v) is
(1)  if ((r mod n) + 1 = i)
(2)    then repeat arw_view_i ← S[r].collect(); FD_VAL[i] ← C.read()
(3)         until (PROP_C(arw_view_i)) end repeat
(4)    else FD_VAL[i] ← C.read()
(5)  end if;
(6)  iis_view ← S[r].arw_write_snapshot(v);
(7)  return (iis_view).

operation C*.read() is return (FD_VAL[i]).
```

Fig. 2. A generic simulation of $\mathcal{IIS}_n[C^*]$ in $\mathcal{ARW}_n[C]$: code for a simulator p_i

Where is the problem to solve. If the underlying model was $\mathcal{ARW}_n[\emptyset]$ (no failure detector), the simulation of the operation $IS[r]$.write_snapshot() would boil down to a simple call to $S[r]$.arw_write_snapshot() (lines 6-7). Hence, the main difficulty to simulate $IS[r]$.write_snapshot(v) comes from the presence of the failure detector C.

This comes from the fact that, in all executions, we need to guarantee a correct association between the schedule of the (simulated) invocations of $IS[r]$.write_snapshot() and the outputs of the simulated failure detector C^*. This, which depends on the output of the underlying failure detector C, requires to appropriately synchronize, at every round r, the simulation of the invocations of $IS[r]$.write_snapshot(). Once this is done, the set-linearization of the simulated invocations of $IS[r]$.write_snapshot() follows from the set-linearization of these invocations in the $\mathcal{ARW}_n[C]$ model.

Associate each round with a simulator. The simulation associates each round r with a simulator (we say that the corresponding simulator "owns" round r) in such a way that each correct simulator owns an infinite number of rounds. This is implemented with a simple round-robin technique (line 1).

Simulation of $IS[r]$.write_snapshot(v). When a simulated process q_i issues the invocation $IS[r]$.write_snapshot(v), the simulator p_i first checks if it is the owner of the corresponding round r. If it is not, it refreshes the value of $FD\_VAL[i]$ (line 4) and executes the "common part", namely, it invokes $S[r]$.arw_write_snapshot(v) (line 6) which returns it a set iis_view constituting the result of the invocation of $IS[r]$.write_snapshot(v).

If the simulator p_i is the owner of the round, it repeatedly reads asynchronously the current value of the implementation object $S[r]$ (that it stores in arw_view_i) and refreshes the value of $FD\_VAL[i]$ (line 2). This **repeat** statement terminates when the

values of arw_view_i it has obtained satisfy some predicate (line 3). This predicate, denoted $\text{PROP}_C()$, which depends on the failure detector class C, encapsulates the generic dimension of the simulation. Then, after it has exited the loop, the simulator p_i executes the "common" part, i.e., lines 6-7. It invokes $S[r].\mathsf{arw\_write\_snapshot}(v)$ which provides it with a view iis_view which is returned as the result of the invocation of $IS[r].\mathsf{write\_snapshot}(v)$.

The fact that, during each round, (a) some code is executed only by the simulator that owns r, (b) some code is executed only by the other simulators and (c) some code is executed by all simulators, realizes the synchronization discussed above that allows for a correct set-linearization of the invocations of $IS[r].\mathsf{write\_snapshot}()$ in $\mathcal{IIS}_n[C^*]$.

Simulation of $C^*.\mathsf{read}()$. When a simulated process q_i wants to read its local failure detector output, its simulator p_i returns it the last value it has read from its local failure detector variable.

To summarize. When simulating $\mathcal{IIS}_n[C^*]$ on top of $\mathcal{ARW}_n[C]$, we have the following: (a) a faulty simulator p_i gives rise to a faulty simulated process q_i and (b) a correct simulator p_i gives rise either to a strongly correct, a weakly correct or a faulty simulated process q_i in $\mathcal{IIS}_n[C^*]$ (this can depend on $\text{PROP}_C()$).

Moreover, whatever C, we have to show that there is at least one correct process in $\mathcal{IIS}_n[C^*]$. This amounts to show that there is a simulator p_i that executes the infinite sequence $\{IS[r].\mathsf{write\_snapshot}()\}_{r\geq 1}$. To that end, we have to show that each object $IS[r]$ is non-blocking (i.e., whatever the round r and the concurrency pattern, at least one invocation of $IS[r].\mathsf{write\_snapshot}()$ terminates). The corresponding proof is given when we consider specific failure detector classes (see below). Then, due to the structure of the IIS model, the very existence of at least one correct process in $\mathcal{IIS}_n[C^*]$ entails the existence of at least one strongly correct process in this model (see the definition of the set $\mathcal{SC}$ in Section 3).

5.2 Instantiating the Simulation

When $C = \Omega_k$, the property $\text{PROP}_C(arw\_view)$ can be instantiated at each simulator p_i as follows:

$$\text{PROP}_{\Omega_k}(arw\_view_i) \;=\; \big(\exists\, \ell \in FD\_VAL[i] :\; (\ell = i \;\vee\; \exists \langle \ell, - \rangle \in arw\_view_i)\big).$$

Let $leaders_i = FD\_VAL[i]$ (the last value of Ω_k read by the simulator p_i). The previous predicate directs the simulator p_i, at each round r it owns, to wait until $i \in leaders_i$ or until it has seen the simulation of $IS[r].\mathsf{write\_snasphot}()$ issued by a simulator q_j such that $j \in leaders_i$.

Theorem 2. *Let A be an algorithm solving a colorless task in the $\mathcal{IIS}_n[\Omega_k^*]$ model. The simulation of A on top of $\mathcal{ARW}_n[\Omega_k]$ where $IS[r].\mathsf{write\_snapshot}()$ and $C^*.\mathsf{read}()$ are implemented by the algorithms described in Figure 2 and the predicate PROP_C is instantiated by PROP_{Ω_k}, produces an execution of A that could have been obtained in $\mathcal{IIS}_n[\Omega_k^*]$. Moreover, there is a one-to-one correspondence between the correct (simulated) processes in $\mathcal{IIS}_n[\Omega_k^*]$ and the correct simulators in $\mathcal{ARW}_n[\Omega_k]$.*

Due to page limitation, the reader is referred to [7] for the instantiations where $C = \diamond P,\ P,\ \Sigma,\ S,\ \diamond S,\ S_x,\ \diamond S_x$.

6 From Wait-Freedom to t-Resilience

Notation. Let $\mathcal{IIS}_{n,t}[C]$ be the extended $\mathcal{IIS}_n[C]$ model in which at least $n-t$ processes are strongly correct, i.e., $|\mathcal{SC}| \geq n-t$ and $|\mathcal{WC}| + |\mathcal{F}| \leq t$. Similarly, let $\mathcal{ARW}_{n,t}[C]$ be the extended $\mathcal{ARW}_n[C]$ model in which at most t processes are faulty.

From $\mathcal{IIS}_{n,t}[C^*]$ to $\mathcal{ARW}_{n,t}[C]$. Theorem 1 has shown that the simulation described in Figure 1 ensures that (a) any strongly correct simulator in IIS gives rise to a correct simulated process in ARW and (b) a weakly or faulty simulator gives rise to a faulty simulated process. It follows that if $|\mathcal{SC}| \geq n-t$ in $\mathcal{IIS}_{n,t}[C^*]$ we have at most t faulty process in the simulated system $\mathcal{ARW}_{n,t}[C]$.

From $\mathcal{ARW}_{n,t}[C]$ to $\mathcal{IIS}_{n,t}[C^*]$. In this direction, the simulation from $\mathcal{ARW}_n[C]$ in $\mathcal{IIS}_n[C^*]$ presented in Figure 2 can be easily adapted in order to simulate $\mathcal{ARW}_{n,t}[C]$ in $\mathcal{IIS}_{n,t}[C^*]$. It is indeed sufficient to replace PROP_C by $\text{PROP}_C \wedge |arw\_view_i| \geq (n-t-1)$ (it is of course assumed that we do not have $|arw\_view_i| \geq (n-t-1) \Rightarrow \neg \text{PROP}_C$). In this way, at every round r it owns, each correct simulator p_i is constrained to wait until at least $n-t-1$ processes have invoked $S[r]$.arw_write_snapshot() before being allowed to invoke its own. The correction of this extended simulation is captured in the following theorem.

Theorem 3. *Let A be an algorithm solving a colorless task in the $\mathcal{IIS}_n[C^*]$ model. For the failure detector classes studied in this paper, The simulation of A on top of $\mathcal{ARW}_n[C]$ where the invocations of $IS[r]$.write_snapshot() and C^*.read() are implemented by the algorithms described in Figure 2 and the predicate PROP_C is replaced by $\text{PROP}_C \wedge |arw\_view_i| \geq (n-t-1)$, produces a correct execution of A in $\mathcal{IIS}_n[C^*]$ in which $n-t$ processes are strongly correct.*

This work has been partially supported by the French ANR project DISPLEXITY devoted to the computability and complexity in distributed computing.

References

1. Borowsky, E., Gafni, E.: Immediate atomic snapshots and fast renaming. In: Proc. 12th ACM PODC, pp. 41–51 (1993)
2. Borowsky, E., Gafni, E.: A simple algorithmically reasoned characterization of wait-free computations. In: Proc. 16th ACM PODC, pp. 189–198 (1997)
3. Chandra, T., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43(2), 225–267 (1996)
4. Herlihy, M.P.: Wait-free synchronization. ACM TOPLAS 13(1), 124–149 (1991)
5. Rajsbaum, S., Raynal, M., Travers, C.: The Iterated Restricted Immediate Snapshot Model. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 487–497. Springer, Heidelberg (2008)
6. Rajsbaum, S., Raynal, M., Travers, C.: An impossibility about failure detectors in the iterated immediate snapshot model. Information Processing Letters 108(3), 160–164 (2008)
7. Raynal, M., Stainer, J.: Increasing the Power of the Iterated Immediate Snapshot Model with Failure Detectors. Tech Report #1991, IRISA, Université de Rennes (F) (2011)

Improved Approximation for Orienting Mixed Graphs*

Iftah Gamzu[1] and Moti Medina[2,**]

[1] Computer Science Division, The Open Univ., and
Blavatnik School of Computer Science, Tel-Aviv Univ., Israel
iftah.gamzu@cs.tau.ac.il
[2] School of Electrical Engineering, Tel-Aviv Univ., Israel
medinamo@eng.tau.ac.il

Abstract. An instance of the maximum mixed graph orientation problem consists of a mixed graph and a collection of source-target vertex pairs. The objective is to orient the undirected edges of the graph so as to maximize the number of pairs that admit a directed source-target path. This problem has recently arisen in the study of biological networks, and it also has applications in communication networks.

In this paper, we identify an interesting local-to-global orientation property. This property enables us to modify the best known algorithms for maximum mixed graph orientation and some of its special structured instances, due to Elberfeld et al. (CPM '11), and obtain improved approximation ratios. We further proceed by developing an algorithm that achieves an even better approximation guarantee for the general setting of the problem. Finally, we study several well-motivated variants of this orientation problem.

1 Introduction

An instance of the *maximum mixed graph orientation* problem consists of a *mixed* graph $G = (V, E_{\mathrm{D}} \cup E_{\mathrm{U}})$ with n vertices, such that E_{D} and E_{U} indicate the sets of directed and undirected edges, respectively. An additional ingredient of the input is a collection $P \subseteq V \times V$ of source-target vertex pairs. A source-target vertex pair $(s, t) \in P$ is called a *request*. The objective is to orient G in a way that maximizes the number of satisfied requests. An *orientation* of G is a directed graph $\boldsymbol{G} = (V, E_{\mathrm{D}} \cup \boldsymbol{E}_{\mathbf{U}})$, where $\boldsymbol{E}_{\mathbf{U}}$ is a set of directed edges obtained by choosing a single direction for each undirected edge in E_{U}. A request (s, t) is said to be *satisfied* under an orientation $\boldsymbol{G}$ if there is a directed path from s to t in $\boldsymbol{G}$.

One may assume without loss of generality that the mixed graph G is *acyclic*, that is, a graph that has no cycles. This assumption holds since any instance of maximum mixed graph orientation can be reduced to another instance in which the underlying mixed graph is acyclic without affecting the number of requests that can be satisfied [16,6].

* Due to space limitations, some proofs are omitted from this extended abstract. We refer the reader to the full version of this paper (available online at http://arxiv.org/abs/1204.0219), in which all missing details are provided.

** M.M was partially funded by the Israeli ministry of Science and Technology.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 243–253, 2012.

Indeed, if the input graph contains cycles, one can sequentially contract them one after the other. In each step, the undirected edges of an arbitrary cycle are all oriented in the same direction. In particular, if this cycle contains directed edges then the undirected edges are oriented in a consistent way with those edges. As a result, every pair of vertices on this cycle admits a directed path between them, and thus, the cycle can be contracted. One can easily validate that the resulting mixed acyclic graph consists of undirected components, each of which must be an undirected tree, and those components are connected by directed edges in a way that does not produce cycles. The maximum mixed graph orientation problem draws its interest from applications in network biology and communication networks:

Network Biology. Recent technological advances, such as yeast two-hybrid assays [8] and protein co-immunoprecipitation screens [11], enable detecting physical interactions in the cell, leading to protein-protein interaction (PPI) networks. One major caveat of those PPI measurements is that they do not reveal information about the directionality of the interactions, namely, the directions in which the signal flows. Since PPI networks serve as the skeletons of signal transduction in the cell, inferring the hidden directionality information may provide insights to the inner working of the cell. Such an information may be inferred from causal relations in those networks [17]. One such source of causal relations is perturbation experiments, in which a gene is perturbed (cause) and as a result, other genes change their expression levels (effects). A change of expression of a gene suggests that the corresponding proteins admit a path in the network, and in particular, it is assumed that there must be a directed path from the causal gene to the affected gene.

Up until this point in time, the above-mentioned scenario can be modeled as a special instance of the maximum mixed graph orientation problem in which one is interested to orient the edges of an *undirected* network in a way that maximizes the number of cause-effect pairs that admit a directed path from the causal gene to the affected gene. However, in the more accurate biological variant, there are several interactions whose directionality is known in advance. For instance, protein-DNA interactions are naturally directed from a transcription factor to its regulated genes, and some PPIs, like kinase-substrate interactions, are known to transmit signals in a directional fashion. Therefore, in general, the input network is a mixed graph.

Communication Networks. A unidirectional communication network consists of communication links that allow data to travel only in one direction. One main benefit of such communication links is that the data of the device on one side is kept confidential while it may still access the data of the device on the other side. As a consequence, unidirectional networks are most commonly found in high security environments, where a connection may be made between devices with differing security classifications. For example, unidirectional communication links can be used to facilitate access to a vulnerable domain such as the Internet to devices storing sensitive data. The maximum mixed graph orientation problem captures the interesting scenario in which one is interested to design a unidirectional network that maximizes the number of connection requests that can be satisfied in a secure way. We remark that unidirectional networks have also been studied in distributed and wireless ad hoc settings (see, e.g., [2,1,14] and the references therein), where a common focus is on algorithmic questions that arise in

a given unidirectional network. Here, we are rather interested in the question of how to design such a network while optimizing some performance guarantees.

1.1 Previous Work

Arkin and Hassin [3] seem to have been the first to study the problem of orienting mixed graphs. They focused on the decision problem corresponding to maximum mixed graph orientation, and demonstrated that it is NP-complete. Elberfeld et al. [6] observed that the reduction in their proof implies that the maximum mixed graph orientation problem is NP-hard to approximate to within a factor of $7/8$. Silverbush, Elberfeld, and Sharan [16] devised a polynomial-size integer linear program formulation for this problem, and evaluated its performance experimentally. Recently, Elberfeld et al. [6] developed several polylogarithmic approximation algorithms for special instances of the problem in which the underlying graph is tree-like, e.g., when the graph has bounded treewidth. In addition, they developed a greedy algorithm for the general setting that achieves $\Omega(1/(M^c \log n))$-approximation, where $M = \max\{n, |P|\}$ and $c = 1/\sqrt{2} \approx 0.7071$.

Medvedovsky et al. [15] initiated the study of the special setting of maximum graph orientation in which the underlying graph is undirected, that is, when there are no pre-directed edges. They proved that it is NP-hard to approximate this problem to within a factor of $12/13$, even when the graph is a star. They also proposed an exact dynamic-programming algorithm for the special case of path graphs, and a $\Omega(1/\log n)$-approximation algorithm for the general problem. Gamzu, Segev and Sharan [10] utilized the framework developed in [9] to obtain an improved $\Omega(\log\log n/\log n)$-approximation ratio (see also [5]). Very recently, Dorn et al. [4] studied this problem from a parameterized complexity point of view. They presented several fixed-parameter tractability results. Further research focused on other variants of this undirected orientation problem. For example, Hakimi, Schmeichel, and Young [12] studied the special setting in which the set of requests contains all vertex pairs, and developed an exact polynomial-time algorithm.

1.2 Our Results

We identify a useful structural property of requests crossing through a junction vertex. Informally, this property guarantees that if a set of requests is locally satisfiable then it can also be satisfied globally. Using this property, we can slightly modify the algorithms developed by Elberfeld et al. [6], and obtain improved approximation ratios. For example, we eliminate a logarithmic factor from their polylogarithmic approximation ratio for the case that the underlying graph has bounded treewidth. These results appear in Section 2. Although the local-to-global property can be used in conjunction with the algorithm of Elberfeld et al. [6] to obtain an improved approximation guarantee for the general setting, we proceed by developing an improved $\Omega(1/(n|P|)^{1/3})$-approximation algorithm for this problem. Our algorithm is based on a greedy approach that employs the local-to-global property in a novel way. The specifics of this algorithm are presented in Section 3. We also study two well-motivated variants of the orientation problem, and most notably, show hardness results for them. Further details are provided in Section 4.

2 From Local to Global Orientations

In this section, we identify a useful structural property of requests crossing through a junction vertex. Informally, this property guarantees that if there is an orientation of the *local* neighborhood of a vertex that locally satisfies a set of requests then it can be extended to a *global* orientation of the complete graph which satisfies the same set of requests. Finding a local orientation that maximizes the local satisfiability is a relatively easy task, namely, it admits a constant factor approximation algorithm. As a consequence, we can slightly modify the algorithms developed by Elberfeld et al. [6] so they utilize this property, and obtain improved approximation ratios. For example, we eliminate a logarithmic factor from their polylogarithmic approximation ratio for the special case that the underlying graph has bounded treewidth.

We associate each request $(s, t) \in P$ with the shortest path p between s and t in the underlying graph. Note that in case there are several shortest paths for a request, we associate it with one of them arbitrarily. We now introduce some notation and terminology. To better understand the suggested notation, we refer the reader to the concrete example in Figure 1.

- The *local neighborhood* of a vertex v is the subgraph G_v that consists of v, all edges incident on v, and all vertices adjacent to v. Notice that the local neighborhood graph is a star.
- Let P_v be the set of shortest paths of requests that cross v, and let P'_v be the corresponding set of *local paths*, that is, the paths of P_v confined to the local neighborhood of v. More precisely, each (global) path $p \in P_v$ gives rise to a (local) path $p' \in P'_v$ defined as the intersection of p with the local neighborhood of v. Furthermore, for each $p' \in P'_v$, we define its local endpoints s' and t' to be the closest vertices to s and t on p that also appear on p', respectively.
- The *local graph orientation* problem corresponding to vertex v is defined with respect to the local neighborhood graph G_v and the set of local paths P'_v. The goal is to orient the undirected edges of G_v in a way that maximizes the number of satisfied paths in P'_v. A path is said to be *satisfied* if there is a directed path between its source and target vertices under the orientation.

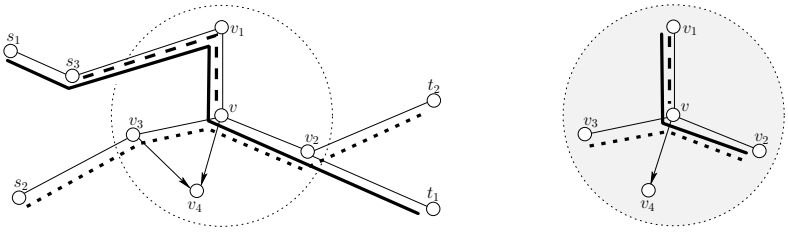


Fig. 1. (a) Suppose $P = \{(s_1, t_1), (s_2, t_2), (s_3, v)\}$ is the set of requests, and note that the shortest paths of these requests are marked with the heavy lines. Notice that all these paths cross v. (b) The local neighborhood of v, and the corresponding set of local paths. For example, notice that the local endpoints of the request (s_1, t_1) are $s'_1 = v_1$ and $t'_1 = v_2$.

Lemma 1. *Given an orientation of G_v that satisfies a set of local paths $S' \subseteq P'_v$ then there is an orientation of G that satisfies the corresponding set of global paths $S \subseteq P_v$.*

Proof. We argue that if two local paths $p'_1, p'_2 \in S'$ then the corresponding global paths $p_1, p_2 \in S$ cannot be in conflict. The paths p_1 and p_2 are said to be *in conflict* if they have a mutual undirected edge that gets a different direction when the edges of p_1 are consistently oriented from its source vertex to its target vertex and when the edges of p_2 are consistently oriented from its source vertex to its target vertex. Notice that establishing this argument completes that proof of the lemma since none of the paths of S can be in conflict with another path in S, and therefore, all the paths in S can be simultaneously satisfied by consistently orienting each one of them from its source vertex to its target vertex. Note that after one orients those paths, the remaining undirected edges of the graph can be oriented in some arbitrary way.

For the purpose of establishing the above argument, let us suppose that p_1 and p_2 are in conflict, and attain a contradiction. Since p_1 and p_2 are in conflict then there is an undirected edge $e = (v_1, v_2) \in E_{\mathrm{U}}$ that gets a different direction when consistently orienting each one of p_1 and p_2 from its source vertex to its target vertex. Let us assume without loss of generality that edge e is the closest to v from all conflicting edges. We next present a case analysis that depends whether the edge e appears before or after the position of vertex v on each of paths p_1 and p_2. Essentially, there are two main cases. To better understand the used notation, we refer the reader to the concrete examples in Figure 2.

Case I: edge e appears after vertex v in both p_1 and p_2. Let us assume without loss of generality that v_1 is closer to v than v_2 on p_1, and v_2 is closer to v than v_1 on p_2. Let d_1 be the distance between v and v_1 on p_1, and d_2 be the distance between v and v_2 on p_2. Since p_1 is a shortest path between s_1 and t_1, it must also be a shortest path between v and v_2. Thus, $d_1 + 1 \leq d_2$. Similarly, since p_2 is a shortest path between s_2 and t_2, it must also be a shortest path between v and v_1, and hence, $d_2 + 1 \leq d_1$. Summing together the above inequalities results in $d_1 + d_2 + 2 \leq d_1 + d_2$, a contradiction.

We note that the case that the edge e appears *before* vertex v in both p_1 and p_2 can be handled along the same lines with an adjustment to the relative position of v, e.g., the distances need to be defined from v_1 and v_2 towards the junction vertex v.

Case II: edge e appears after vertex v in p_1 and before vertex v in p_2. Let us assume without loss of generality that v_1 is closer to v than v_2 on both paths p_1 and p_2. Since $p'_1, p'_2 \in S'$ we know that the edge on which p_1 leaves v and the edge on which p_2 enters v must be different. This implies that the subpath between v and v_1 on p_1 and the subpath between v_1 and v on p_2 are different. Consequently, merging these two subpaths creates a cycle in the graph. This contradicts the fact that the graph is acyclic.

Note that the case that the edge e appears after vertex v in p_2 and before vertex v in p_1 is essentially identical to the above case up to a renaming of the paths. ■

We now concentrate on the computational complexity of the local graph orientation problem corresponding to a vertex v. One can easily validate that this problem is equivalent to the maximum undirected graph orientation problem on a star. Medvedovsky et al. [15] demonstrated that this problem is equivalent to the maximum directed cut

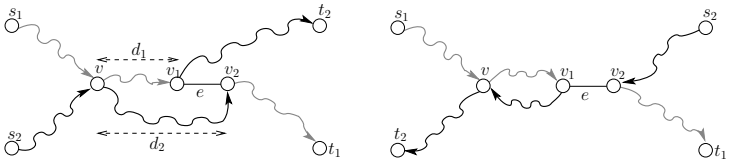


Fig. 2. (a) The case that e appears after v in both p_1 and p_2. (b) The case that e appears after v in p_1 and before v in p_2.

problem. This latter problem admits constant factor approximation algorithms (see, e.g., [7,13]). In fact, one can easily verify that a random orientation of the undirected edges in the local neighborhood satisfies at least $1/4$ of the paths of P'_v in expectation. This follows since the maximal length of any path in the local neighborhood is at most 2. Furthermore, one can use the method of conditional expectations to obtain a deterministic orientation that satisfies at least $1/4$ of the paths, and consequently, this approach is a $1/4$-approximation for this problem. Combining this result with the local-to-global orientation property exhibited in Lemma 1 implies the following theorem.

Theorem 1. *Given a vertex v and a set of requests P_v whose shortest paths cross v, there is a polynomial-time algorithm that computes an orientation that satisfies $\Omega(|P_v|)$ requests.*

We can now modify the algorithms developed by Elberfeld et al. [6] in accordance with Theorem 1, and obtain the following improved approximation ratios. We emphasize that the algorithms and their analysis follow (up to our modification step) those presented by Elberfeld et al. [6], and thus, we defer them to the full version of the paper. The first two theorems present algorithms whose approximation guarantees depend on the treewidth and feedback vertex number of the underlying graph.

Theorem 2. *There is a polynomial-time algorithm that finds an orientation satisfying $\Omega(|P|/(k \log n))$ requests when the undirected version of the underlying graph has bounded treewidth k.*

Theorem 3. *There is a polynomial-time algorithm that finds an orientation satisfying $\Omega(|P|/(k+\log n))$ requests, where k is the minimum number of vertices whose deletion turns the undirected version of the underlying graph into a tree.*

We can also improve the approximation ratios of the algorithms presented by Elberfeld et al. [6] for the general case, in which there are no structural assumptions on the graph, by a logarithmic factor.

Theorem 4. *There is a polynomial-time algorithm that approximates the maximum mixed graph orientation problem to within a factor of $\Omega(1/\sqrt{\Delta|P|})$, where Δ is the maximum length of a shortest source-target path in the graph.*

Theorem 5. *There is a polynomial-time algorithm that approximates the maximum mixed graph orientation problem to within a factor of* $\Omega(1/M^{1/\sqrt{2}})$*, where* $M = \max\{n, |P|\}$.

Note that we do not provide a proof for the latter theorem since it can be established along the same lines of [6], but more importantly, since we next present an algorithm with a better approximation guarantee.

3 Improved Approximation for the General Case

In this section, we develop a relatively simple $\Omega(1/(n|P|)^{1/3})$-approximation algorithm for the maximum mixed graph orientation problem. Our algorithm is based on a greedy approach that employs the local-to-global orientation property developed in Section 2.

The algorithm, formally described below, begins by associating each request $(s_i, t_i) \in P$ with a shortest path p_i between s_i and t_i in the graph. Then, it greedily orients shortest paths one after the other until all the remaining paths are in conflict with many other paths. When this happens, the algorithm concentrates on the vertex that is crossed by a maximal number of paths, and utilizes the local-to-global orientation algorithm from Theorem 1 to complete the orientation of the graph. Recall that two paths p_1 and p_2 are said to be *in conflict* if they have a mutual undirected edge that gets a different direction when the edges of p_1 and p_2 are consistently oriented from their source vertex to their target vertex.

Algorithm 1. Greedy Orientation

Input: A mixed graph G and a collection $P \subseteq V \times V$ of requests
Output: An orientation $\boldsymbol{G}$ of G

1: Let p_i be a shortest path for request $(s_i, t_i) \in P$ in G, and let $\mathcal{P} = \bigcup\{p_i\}$
2: **while** there is $p_i \in \mathcal{P}$ that is in conflict with less than $(n|P|)^{1/3}$ paths in $\mathcal{P}$ **do**
3: Let $\mathcal{Q} \subseteq \mathcal{P}$ be the set of paths in conflict with p_i
4: $G \leftarrow$ the graph that results by orienting the edges of p_i from s_i towards t_i in G
5: $\mathcal{P} \leftarrow \mathcal{P} \setminus (\mathcal{Q} \cup \{p_i\})$
6: **end while**
7: Let v be a vertex that a maximal number of paths in $\mathcal{P}$ cross, and let $\mathcal{P}_v \subseteq \mathcal{P}$ be that set of paths
8: $G \leftarrow$ the graph that results by executing the algorithm from Theorem 1 with respect to v and $\mathcal{P}_v$
9: **return** G

One can easily verify that the algorithm computes a feasible orientation, namely, it assigns a single direction to each undirected edge. This follows since no conflicting paths are oriented during the main loop of the algorithm, and since the algorithm from Theorem 1 is known to compute a feasible orientation. We next prove that the algorithm satisfies $\Omega(1/(n|P|)^{1/3})$-fraction of all requests. Clearly, this implies that the algorithm achieves (at least) the same approximation guarantee.

Theorem 6. *The greedy orientation algorithm satisfies $\Omega(1/(n|P|)^{1/3})$-fraction of all requests.*

Proof. Let $\mathcal{P} = \bigcup\{p_i\}$ be the initial collection of shortest paths, and note that $|\mathcal{P}| = |P|$. In addition, let $\mathcal{P}_2 \subseteq \mathcal{P}$ be the set of paths the remain after the termination of the main loop of the algorithm, and $\mathcal{P}_1 = \mathcal{P} \setminus \mathcal{P}_2$. Finally, let $\mathcal{A}_1$ be the set of paths that our algorithm satisfies during the main loop of the algorithm, and let $\mathcal{A}_2$ be the set of paths that the algorithm satisfies during the execution of the algorithm from Theorem 1. In what follows, we prove that $|\mathcal{A}_1| = \Omega(1/(n|P|)^{1/3}) \cdot |\mathcal{P}_1|$, and $|\mathcal{A}_2| = \Omega(1/(n|P|)^{1/3}) \cdot |\mathcal{P}_2|$. Consequently, we obtain that the number of paths satisfied by our algorithm is

$$|\mathcal{A}_1| + |\mathcal{A}_2| = \Omega\left(\frac{1}{(n|P|)^{1/3}}\right) \cdot (|\mathcal{P}_1| + |\mathcal{P}_2|) = \Omega\left(\frac{1}{(n|P|)^{1/3}}\right) \cdot |P| \,.$$

The fact that $|\mathcal{A}_1| = \Omega(1/(n|P|)^{1/3}) \cdot |\mathcal{P}_1|$ easily follows by observing that in each step of the main loop of the algorithm, one path is satisfied while less than $(n|P|)^{1/3}$ paths are discarded. Hence, we are left to prove that $|\mathcal{A}_2| = \Omega(1/(n|P|)^{1/3}) \cdot |\mathcal{P}_2|$. We establish a somewhat stronger result by demonstrating that $|\mathcal{A}_2| = \Omega(1/(n|\mathcal{P}_2|)^{1/3}) \cdot |\mathcal{P}_2|$. For this purpose, consider two paths $p_1, p_2 \in \mathcal{P}_2$ that are in conflict. We associate the conflict between these paths to an arbitrary undirected edge that gets a different direction when p_1 and p_2 are oriented, and place one token on this edge. Notice that each path of $\mathcal{P}_2$ is in conflict with at least $(n|P|)^{1/3}$ other paths in $\mathcal{P}_2$; otherwise, the main loop would not have terminated. This implies that if we place a token for each pair of conflicting paths in $\mathcal{P}_2$ as shown before then the undirected edges of G have at least $(n|P|)^{1/3} \cdot |\mathcal{P}_2|/2 \geq n^{1/3}|\mathcal{P}_2|^{4/3}/2$ tokens placed on them. As a consequence, there must be a vertex that has at least $t = |\mathcal{P}_2|^{4/3}/(2n^{2/3})$ tokens placed on the undirected edges in its local neighborhood. We next argue that if some vertex has t tokens in its local neighborhood then there must be $\Omega(\sqrt{t})$ paths that cross that vertex. As a result, we attain that the number of paths that cross the vertex v, i.e., the vertex that a maximal number of paths from $\mathcal{P}_2$ cross, is at least $\Omega(\sqrt{t}) = \Omega(|\mathcal{P}_2|^{2/3}/n^{1/3})$. By theorem 1, our algorithm satisfies a constant fraction of these requests, namely, $|\mathcal{A}_2| = \Omega(1/(n|\mathcal{P}_2|)^{1/3}) \cdot |\mathcal{P}_2|$, as required.

For the purpose of establishing the above argument, consider some vertex u that has t tokens in its local neighborhood. Let us focus on some edge e in this local neighborhood that has r paths that traverse in one direction and ℓ paths that traverse in the other direction. Notice that such an edge is assigned $r \cdot \ell$ tokens. This implies that if the local neighborhood of u consists only of the edge e then the minimal number of paths that cross u corresponds to the solution of $\min\{r + \ell : r \cdot \ell = t\}$. One can easily verify that the solution for this expression is $r = \ell = \sqrt{t}$, that is, the number of paths is $\Omega(\sqrt{t})$. Note that when there is more than one edge in the local neighborhood of u then any path may cross at most two edges. As a result, if we denote the set of edges in the local neighborhood of u by E_u, then the minimal number of paths that cross u dominates the solution of $\min\{\sum_{e \in E_u}(r_e + \ell_e)/2 : \sum_e (r_e \cdot \ell_e) = t\}$; here, r_e and ℓ_e indicate the number of paths traversing edge e in one direction and the other direction, respectively. One can easily demonstrate that the solution for the above expression is obtained by

assigning non-zero values only to one pair of r_e, ℓ_e variables, namely, it is equivalent to the solution for the single edge case. ■

4 Other Orientation Variants

In this section, we study two well-motivated variants of the orientation problem: the first is maximum mixed graph orientation *with fixed paths*, and the other is maximum mixed *grid* orientation.

4.1 Orientation with Fixed Paths

We consider the maximum mixed graph orientation *with fixed paths* problem. This variant is identical to the maximum mixed graph orientation problem with the exception that each request $(s,t) \in P$ is also associated with a fixed path p from s to t in the graph. With this modified definition in mind, a request (s,t) is satisfied only if the edges of the path p are oriented from the vertex s towards the vertex t. Note that this variant is seemingly simpler than maximum mixed graph orientation since the only computational task is to decide which requests to satisfy, and there is no need to decide which paths will be used to satisfy those requests. This is also one of our motivations for studying this variant, hoping that it will shed some light on the original problem that would lead to a reduction in the gap between its lower and upper approximation bounds.

We prove that the maximum mixed graph orientation with fixed paths problem is NP-hard to approximate to within a factor of $\max\{1/|P|^{1-\epsilon}, 1/m^{1/2-\epsilon}\}$, for any $\epsilon > 0$. In fact, we establish this result even when the underlying graph is undirected. As a consequence, we attain that this problem is provably harder than the maximum mixed (or undirected) graph orientation problem, although it may seem simpler at first glance. Our proof is based on showing that the problem under consideration captures the well-known *maximum independent set* problem as a special case.

Theorem 7. *The maximum mixed graph orientation with fixed paths problem is NP-hard to approximate within a factor of* $\max\{1/|P|^{1-\epsilon}, 1/m^{1/2-\epsilon}\}$*, for any* $\epsilon > 0$.

4.2 Orientation in Grid Networks

We study the maximum mixed *grid* orientation problem. This variant is identical to the maximum mixed graph orientation problem with the additional restriction that the graph is a grid. A $n \times m$ *grid* network is a graph with a vertex set $V = \{1,\ldots,n\} \times \{1,\ldots,m\}$, and an edge set E consisting of horizontal edges, i.e., edges $((i,j),(i,j+1))$ for all $j = \{1,\ldots,m-1\}$, and vertical edges, i.e., edges $((i,j),(i+1,j))$ for all $i = \{1,\ldots,n-1\}$. Note that the study of this variant is motivated by applications in networking.

We prove that the maximum mixed grid orientation problem is at least as hard as the *maximum directed cut* problem. Consequently, approximating our problem within factors of $12/13 \approx 0.923$ and $\alpha_{\mathrm{GW}} \approx 0.878$ is NP-hard and Unique Game-hard, respectively. Interestingly, this finding comes in contrast with the results attainable for

the undirected grid setting. This latter setting can be solved to optimality in polynomial-time, and in particular, when the grid is not a path, that is, when $n, m > 1$, all the requests in P can be satisfied.

Theorem 8. *The maximum mixed grid orientation problem is NP-hard to approximate within a factor of* $12/13 \approx 0.923$*, and Unique Games-hard to approximate within a factor of* $\alpha_{\mathrm{GW}} \approx 0.878$.

Orientation of Undirected Grids. The above-mentioned hardness result comes in contrast with the results attainable for the undirected grid setting. This latter setting can be solved to optimality in polynomial-time. Specifically, when the grid is a path, i.e., when either m or n equals 1, there are optimal polynomial-time algorithms for the problem [15,4], and when $n, m > 1$, there is a simple orientation that satisfies all the requests in P. This orientation can be obtained by creating a directed cycle along the perimeter of the grid, and then, orienting all the remaining horizontal and vertical edges consistently.

References

1. Afek, Y., Bremler-Barr, A.: Self-stabilizing unidirectional network algorithms by power supply. Chicago J. Theor. Comput. Sci. (1998)
2. Afek, Y., Gafni, E.: Distributed algorithms for unidirectional networks. SIAM J. Comput. 23(6), 1152–1178 (1994)
3. Arkin, E.M., Hassin, R.: A note on orientations of mixed graphs. Discrete Applied Mathematics 116(3), 271–278 (2002)
4. Dorn, B., Hüffner, F., Krüger, D., Niedermeier, R., Uhlmann, J.: Exploiting Bounded Signal Flow for Graph Orientation Based on Cause–Effect Pairs. In: Marchetti-Spaccamela, A., Segal, M. (eds.) TAPAS 2011. LNCS, vol. 6595, pp. 104–115. Springer, Heidelberg (2011)
5. Elberfeld, M., Bafna, V., Gamzu, I., Medvedovsky, A., Segev, D., Silverbush, D., Zwick, U., Sharan, R.: On the approximability of reachability-preserving network orientations. Internet Mathematics 7, 209–232 (2011)
6. Elberfeld, M., Segev, D., Davidson, C.R., Silverbush, D., Sharan, R.: Approximation Algorithms for Orienting Mixed Graphs. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 416–428. Springer, Heidelberg (2011)
7. Feige, U., Goemans, M.X.: Aproximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In: 3rd ISTCS, pp. 182–189 (1995)
8. Fields, S.: High-throughput two-hybrid analysis: The promise and the peril. The FEBS Journal 272(21), 5391–5399 (2005)
9. Gamzu, I., Segev, D.: A Sublogarithmic Approximation for Highway and Tollbooth Pricing. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 582–593. Springer, Heidelberg (2010)
10. Gamzu, I., Segev, D., Sharan, R.: Improved Orientations of Physical Networks. In: Moulton, V., Singh, M. (eds.) WABI 2010. LNCS, vol. 6293, pp. 215–225. Springer, Heidelberg (2010)
11. Gavin, A.C., Bosche, M., Krause, R., Grandi, P., Marzioch, M., Bauer, A., Schultz, J., Rick, J.M., Michon, A.M., Cruciat, C.M., Remor, M., Hofert, C., Schelder, M., Brajenovic, M., Ruffner, H., Merino, A., Klein, K., Hudak, M., Dickson, D., Rudi, T., Gnau, V., Bauch, A., Bastuck, S., Huhse, B., Leutwein, C., Heurtier, M.A., Copley, R.R., Edelmann, A., Querfurth, E., Rybin, V., Drewes, G., Raida, M., Bouwmeester, T., Bork, P., Seraphin, B., Kuster, B., Neubauer, G., Superti-Furga, G.: Functional organization of the yeast proteome by systematic analysis of protein complexes. Nature 415, 141–147 (2002)

12. Louis Hakimi, S., Schmeichel, E.F., Young, N.E.: Orienting graphs to optimize reachability. IPL 63(5), 229–235 (1997)
13. Lewin, M., Livnat, D., Zwick, U.: Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 67–82. Springer, Heidelberg (2002)
14. Marina, M.K., Das, S.R.: Routing performance in the presence of unidirectional links in multihop wireless networks. In: 3rd MobiHoc, pp. 12–23 (2002)
15. Medvedovsky, A., Bafna, V., Zwick, U., Sharan, R.: An Algorithm for Orienting Graphs Based on Cause-Effect Pairs and Its Applications to Orienting Protein Networks. In: Crandall, K.A., Lagergren, J. (eds.) WABI 2008. LNCS (LNBI), vol. 5251, pp. 222–232. Springer, Heidelberg (2008)
16. Silverbush, D., Elberfeld, M., Sharan, R.: Optimally Orienting Physical Networks. In: Bafna, V., Sahinalp, S.C. (eds.) RECOMB 2011. LNCS, vol. 6577, pp. 424–436. Springer, Heidelberg (2011)
17. Yeang, C.H., Ideker, T., Jaakkola, T.: Physical network models. Journal of Computational Biology 11(2/3), 243–262 (2004)

Analysis of Random Walks Using Tabu Lists*

Karine Altisen, Stéphane Devismes, Antoine Gerbaud, and Pascal Lafourcade

Grenoble Université, VERIMAG, France

Abstract. A tabu random walk on a graph is a partially self-avoiding random walk which uses a bounded memory to avoid cycles. This memory is called a tabu list and contains vertices already visited by the walker. The size of the tabu list being bounded, the way vertices are inserted and removed from the list, called here an update rule, has an important impact on the performance of the walk, namely the mean hitting time between two given vertices.

We define a large class of tabu random walks, characterized by their update rules. We enunciate a necessary and sufficient condition on these update rules that ensures the finiteness of the mean hitting time of their associated walk on every finite and connected graph. According to the memory allocated to the tabu list, we characterize the update rules which yield smallest mean hitting times on a large class of graphs. Finally, we compare the performances of three collections of classical update rules according to the size of their associated tabu list.

1 Introduction

A *random walk* is a mathematical formalization of a route taken by a walker through a topology of locations: at each step, the next destination is randomly chosen. Random walks are commonly used to model phenomena from many fields like physics or economics. Random walks are inherently distributed algorithms which do not need any knowledge except the list of next possible destinations. Random walks are probabilistic algorithms and this is one of their main advantages. Indeed, since they are non-deterministic (and thus non-predictable), they can be used to build *resilient* algorithms in a fault prone environment or facing intruders [1,2]. Even if there is a change in the environment, *e.g.*, in the topology, or a problem due to an intruder or a failure, then a resilient algorithm still ensures a certain quality of service.

The main drawback of random walks is the large number of steps generally needed to reach one vertex starting from another one, namely the *hitting time*. This is mainly due to the fact that the walker may come back to previously visited vertices, forming loops. We study *partially self-avoiding* random walks on *finite* graphs. They are variants of the *simple random walk*, for which at each step the walker chooses its next destination uniformly at random among all its neighbors.

We add a bounded memory to the walker in order to reduce the number of loops. This memory is called a *tabu list* and contains a part of already visited vertices. We say that a tabu list is of length m, if the list can contain at most m elements. The walker avoids every vertex contained in its tabu list unless it has no choice. One can expect that the tabu list helps to reduce the mean hitting time of the walk.

* This work has been partially supported by ANR Project Aresa2 and MSTIC Project Terra.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 254–266, 2012.

As the size of the tabu list is bounded, when the list is full, one element has to be removed before any new insertion. We call *update rule* the algorithm which drives the policy of insertion and removal in the list. For example, in the $FIFO_m$ update rule, the tabu list is of length m, the current position of the walker is always inserted, and when the list is full, the oldest element is firstly removed to make room for the current position. In particular, $FIFO_1$ yields a *non-backtracking* random walk: the walker never backtracks to the last visited vertex unless it is the only neighbor of the vertex currently visited.

Contribution. A tabu random walk is characterized by its update rule. We define a large class of update rules and analytically study their associated tabu random walks. We compare all these tabu random walks *w.r.t.* the mean hitting time between every two given vertices. Mainly, we try to figure out how to handle the memory of the tabu list and what is the best way to do so. The panel of answers we propose here allows to help the choice of an update rule. More precisely, our contribution is threefold:

1. We provide a necessary and sufficient condition on our update rules that ensures the finiteness of the mean hitting time on every graph.
2. We partially answer to the question "What is the best update rule?" by exhibiting a large collection of graphs indexed by a positive integer m, called m-*free* graphs, in which $FIFO_m$ is the optimal (*w.r.t.* the mean hitting time) update rule, provided that the length of the tabu list is at most m. In particular, the 1-*free* class contains all graphs. Therefore, $FIFO_1$ is the optimal policy on every graph if the tabu list contains at most one element.
3. We compare the performances of three collections of classical update rules, *i.e.*, $FIFO_m$, $RAND_m$, and LRU_m, according to the length m of the tabu list. Our results show that no general answer can be given. For some classes of topologies, the mean hitting time decreases when the size of the memory increases. But, counter-intuitively, there exist cases where having more memory is a penalty: We exhibit topologies where the mean hitting time increases when the length of the tabu list increases.
   A important (perhaps surprising) consequence of our results is that for every update rule $FIFO_m$ with $m \geqslant 2$, there exists a graph and two vertices x, y such that the mean hitting time from x to y using $FIFO_m$ is strictly greater than that of the simple random walk. By contrast, $FIFO_1$ always yields smaller mean hitting times than the simple random walk.

Related Work. For a general account on *simple random walks*, we refer to the survey [3] and the forthcoming book [4].

Random walks with memories have received much less attention. Most analytical results deal with infinite graphs, which are irrelevant for our purposes. For example, there are results on *self-avoiding random walks* for infinite graphs, see the survey [5].

On finite graphs, [6] and [7] study random walks that attach memories on the vertices of the graph. Nevertheless, no theoretical analysis of these solutions are yet available.

In [8], the authors study *non-backtracking random walk* on finite and infinite connected graphs with minimum degree two. In particular, they show that for each finite

graph, except cycles, $FIFO_1$ is irreducible. Consequently, the mean hitting time of $FIFO_1$ on these graphs is also finite. Our first result is more general than this latter assertion because it deals with all finite connected graphs and a class of update rules that includes $FIFO_1$.

Outline. The description of a random walk equipped with a tabu list is given in Section 2. Section 3, 4 and 5 describe the three main contributions of this paper. Section 6 gives concluding remarks and perspectives.

Due to the lack of space, definitions are intuitively described and only sketches of proof are provided. All formal definitions and proofs are given in the technical report [9].

2 Tabu Random Walks and Update Rules

In our framework, the walker evolves on a simple, undirected and connected graph. Besides, we assume that the vertex set is *finite* and contains at least two vertices.

2.1 Tabu Random Walks

A *tabu random walk* on a simple graph is a partially self-avoiding random walk, where the walker is endowed with a finite memory and can jump from a node to another, provided that they are neighbors. The memory of the walker, called *tabu list*, contains a part of the vertices already visited by the walker. At step n, the position of the walker is represented by the random variable X_n and the current tabu list by the random variable T_n. We will denote by T_n^i the i-th element of T_n. The successive ordered pairs $(X_n, T_n)_{n\geqslant 0}$ is a Markov chain, called *tabu chain*. The tabu random walk is the sequence $(X_n)_{n\geqslant 0}$ of the successive positions of the walker.

At each step, the walker avoids to revisit vertices which are present in the current tabu list, unless he is forced to. More precisely, for every non-negative integer n, the next visited vertex X_{n+1} is uniform on the set formed by the neighbors of the current vertex X_n which are not in the tabu list T_n. If this is not possible because all neighbors of X_n are already in the tabu list T_n, then the next visited vertex X_{n+1} is uniform on the neighborhood of the current vertex X_n. Afterward, the next tabu list T_{n+1} is obtained using an *update rule*.

2.2 Update Rules

The policy to insert or remove occurrences of vertices in the tabu list is called the *update rule* and denoted by R_m, where m is a parameter that gives the maximum number of elements m in the tabu list, that is its length. By extension, m also called the *length of the update rule*. When the dependence on the update rule R_m needs to be emphasized, we will denote $(X_n(R_m), T_n(R_m))_{n\geqslant 0}$ the associated tabu chain.

Every update rule works as follows:

(1) First, concatenate the current vertex X_n and the current tabu list T_n, the concatenation being noted $X_n \cdot T_n$.
(2) Then, possibly remove an element of $X_n \cdot T_n$.

Note that according to (2), for some policies, the first element of the concatenation $X_n \cdot T_n$ might be directly discarded, which means that the current vertex X_n is actually not inserted in the tabu list, *i.e.*, $T_n = T_{n+1}$.

The formal definitions of a tabu random walk and of an update rule are given in [9]. Below, we describe the construction of the tabu list according to a given update rule R_m. A tabu list is said to be *full* if its length is m. At step n, *if the current tabu list* T_n *is not full* then all elements of T_n distinct from the current vertex X_n are kept, and one of the three disjoint cases occurs:

1. X_n is not inserted and all its occurrences in T_n are kept: $T_{n+1} = T_n$.
2. X_n is inserted and all its occurrences in T_n are kept: $T_{n+1} = X_n \cdot T_n$.
3. X_n is inserted and one of its occurrences in T_n is removed: there exists a random integer C_{n+1} in $\{i \in \{2, \ldots, |T_n| + 1\} : T_n^{i-1} = X_n\}$ such that $T_{n+1} = X_n \cdot T_n^1 \cdots T_n^{C_{n+1}-2} \cdot T_n^{C_{n+1}} \cdots T_n^{|T_n|}$. Besides, the law of the random variable C_{n+1}, conditionally on (X_n, T_n), is fixed by the update rule.

If the tabu list T_n *is full*, then either case 1 occurs or one element is removed from T_n before X_n is inserted. Formally, there exists a random integer C_{n+1} in $\{1, \ldots, m+1\}$ such that

$$T_{n+1} = \begin{cases} T_n & \text{if } C_{n+1} = 1\,, \\ X_n \cdot T_n^1 \cdots T_n^{C_{n+1}-2} \cdot T_n^{C_{n+1}} \cdots T_n^m & \text{if } C_{n+1} \in \{2, \ldots, m+1\}\,. \end{cases}$$

Similarly, the law of the random variable C_{n+1}, conditionally on (X_n, T_n), is fixed by the update rule. Note that when the tabu list is full, case 2 is forbidden in order to ensure that the length of T_{n+1} still remains less than or equal to m: X_n cannot be inserted in T_n without removing any element of T_n.

We highlight the fact that the law of the new tabu list only depends on the occurrences of the current vertex in the current tabu list: given these occurrences, the labels of the vertices does not matter. In other words, we exclude from our study the update rules that explicitly use the value of the labels, *e.g.*, a rule which has a special case for a vertex with label "1".

An update rule is *trivial* if the current vertex is never inserted in the tabu list when the tabu list contains no element. For example, the unique update rule with zero length is trivial. For every trivial update rule, if the tabu list is initially empty, then it remains empty forever and the walker performs a *simple random walk*: at each step, the next visited vertex is chosen uniformly at random among the neighbors of the current vertex.

2.3 Examples of Update Rules

For every non-negative integer m, we describe three update rules $FIFO_m$, LRU_m and $RAND_m$ of length m by giving for every non-negative integer n, the law of the next tabu list T_{n+1} conditionally on (X_n, T_n):[1]

$FIFO_m$: The current vertex X_n is inserted at the beginning (left) of the current tabu list T_n. If T_n was already full, then its rightmost element is firstly removed.

[1] These rules match the requirements given in Subsection 2.2, see [9] for their formal definition.

LRU_m**:** All occurrences of the current vertex X_n in T_n are removed. If T_n is still full afterward, then its rightmost element is removed. Then, X_n is inserted at the beginning (left) of T_n.

$RAND_m$**:** If the current vertex X_n has an occurrence in T_n, then $T_{n+1} = T_n$. Otherwise, if T_n is full, then T_{n+1} is formed by removing one of the $m+1$ elements of $X_n \cdot T_n$ uniformly at random. If T_n is not full, then $T_{n+1} = X_n \cdot T_n$.

Remark that $FIFO_0$, LRU_0 and $RAND_0$ denote the unique trivial update rule with length 0. Note also that when m equals 1 or 2, the update rules $FIFO_m$ and LRU_m coincide and are both distinct from $RAND_m$. However, for every integer $m \geqslant 3$, $FIFO_m$, LRU_m and $RAND_m$ are distinct.

In general, a random walk equipped with a tabu list is not a Markovian process. However, when using $FIFO_m$ (for some positive integer m), the next visited vertex only depends on the current one and on the m previous steps: this is called a *Markov chain with internal states*; refer to [10, p. 177].

3 Finite Mean Hitting Times

For every update rule R_m of length m, the *hitting time* $H_y(R_m)$ of every vertex y is the random number of steps needed by a walker to reach y. It is defined as the first instant when the tabu random walk $(X_n(R_m))_{n\geqslant 0}$ reaches y: $H_y(R_m) = \inf\{n \geqslant 0 : X_n(R_m) = y\}$. The *mean hitting time* $E_{(x,\varepsilon)}H_y(R_m)$ is the mean hitting time of y when the walker starts at x with an empty tabu list ε. Our goal is to characterize the class of update rules that have finite mean hitting times, for all graphs and all vertices x and y.

Definition 1. *We define the two following conditions:*

$(\mathbf{C_1})$ *For every tabu list t and every position of the walker x, if t is not full and does not contain x, then applying the update rule on x and t results in inserting x in t with a positive probability.*

$(\mathbf{C_2})$ *For every tabu list t and every position of the walker x, if t is full and does not contain x, then applying the update rule on x and t results in removing rightmost element from t with a positive probability.*

The conjunction of $(\mathbf{C_1})$ and $(\mathbf{C_2})$ is a necessary and sufficient condition that ensures the finiteness of all mean hitting times for every associated tabu random walk on every graph. The update rules $FIFO_m$, LRU_m and $RAND_m$ satisfy both $(\mathbf{C_1})$ and $(\mathbf{C_2})$. On the contrary, an update rule of length 1 that keeps the unique element of the tabu list until the corresponding vertex is visited again does not satisfy $(\mathbf{C_2})$. Using such an update rule may lead to an infinite mean hitting time. Indeed, on the flower graph F_1 given in Figure 1,[2] if the walker starts at vertex 1 with the empty tabu list and does not hit vertex 0 at his first step, then its tabu list is 1 forever. Thus, the walker never comes back to vertex 1 and, consequently, the walker will never reach vertex 0. Hence, the mean hitting time to reach 0 from 1 is infinite.

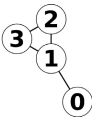


Fig. 1. The flower F_1

[2] F_ℓ is defined in Section 5, for all values of ℓ.

Theorem 2. *Let R_m be an update rule of length m. The mean hitting time $E_{(x,\varepsilon)} H_y(R_m)$ is finite on all graphs, for all vertices x and y, if and only if R_m is either trivial or satisfies* $(\mathbf{C_1})$ *and* $(\mathbf{C_2})$.

Proof. $(\Rightarrow)$ We proceed by establishing the contrapositive. Consider a non trivial update rule that either does not satisfy $(\mathbf{C_1})$ or does not satisfy $(\mathbf{C_2})$. Consider the graph with vertex set $\{0, \ldots, m+2\}$ such that the vertices $1, \ldots, m+2$ form a clique and the vertex 0 has the vertex 1 as unique neighbor. We assume that the walker starts at vertex 1 and does not hit the vertex 0 at its first step. Since the update rule is not trivial, the vertex 1 is inserted in the tabu list with positive probability. Assume that this latter event is realized. Then, the walker needs to return to the vertex 1 before hitting the vertex 0. Since all neighbors of the vertex 1 distinct from 0 have degree $m+1$, the removal of the vertex 1 is needed in order to hit the vertex 0. We now show that the vertex 1 stays forever in the tabu list.

- First, note that with $m = 1$, $(\mathbf{C_1})$ is satisfied, since the update rule is not trivial and when the tabu list is not full, it is empty. So, we need only to consider the case where $m > 1$ and that $(\mathbf{C_1})$ is not satisfied. In this case, we can show that with positive probability, the walker can reach a position with a non-full tabu list from which the tabu list will remain almost surely constant: this tabu list will never be full. Consequently, in this scenario, we cannot remove any element in the tabu list, because the only way to do so requires the tabu list to be full.
- Assume that $(\mathbf{C_1})$ is satisfied, but not $(\mathbf{C_2})$. Similarly, we can show that with positive probability, the walker can reach the position $m+1$ with the full tabu list $m, m-1, \ldots 1$ (the rightmost element is 1). From this configuration, since $(\mathbf{C_2})$ is false, the rightmost element of the tabu list (vertex 1) will never be removed from the list.

Hence, in both cases, with positive probability, the hitting time of vertex 0 is infinite. This implies that the mean hitting time of vertex 0 is infinite.

$(\Leftarrow)$ Suppose first that the update rule is trivial. The walker then performs a simple random walk. Since the simple random walk on each finite and connected graph is positive recurrent, all mean hitting times are finite.

Assume now that the update rule is not trivial and satisfies both $(\mathbf{C_1})$ and $(\mathbf{C_2})$. Consider a graph G. An *essential communicating class* (see [11, p. 16]) for the corresponding tabu chain[3] is a set of ordered pairs formed by a vertex and a tabu list such that the tabu chain cannot exit from and visits each of its elements infinitely often. Let x and y be two vertices of G. Starting from x with empty tabu list, the walker reaches an essential communicating class of the tabu chain in mean finite time. Hence, it suffices to show that the mean hitting time to y is finite, starting from some state of an essential communicating class. Therefore, we assume now that the tabu chain starts from a state (z, t) of a communicating class $\mathcal{C}$. (*N.b.*, starting the walk from (z, t), t may not be empty despite all vertices in t have never been visited.)

[3] Recall that the tabu chain is the Markov chain $(X_n, T_n)_{n \geqslant 0}$.

The restriction of the tabu chain to $\mathcal{C}$ is a *positive recurrent Markov chain* (see [11, p. 11]). So it suffices to show that there exists a tabu list s such that (y, s) belongs to $\mathcal{C}$. Besides, since G is connected, we may assume that y is a neighbor of z. We proceed by contradiction by assuming that there exists one neighbor w of z that will never be reached by the walker. As (z, t) is in $\mathcal{C}$, z is visited infinitely often, so w must have an occurrence in t. Besides, there exists a neighbor of z, w', that does not have any occurrences in t (otherwise t contains all the neighbors of z and there is a positive probability that the walker reaches w). So, when located at z with tabu list t, the walker reaches w' with positive probability. If this event is realized, then we obtain a state (w', u), where u does not contain any occurrence of w'. Moreover, as (z, t) is in $\mathcal{C}$, (w', u) is also in $\mathcal{C}$. Assume now that the walker is in w' with tabu list u. We distinguish between two cases:

- First, assume that u is not full. According to $(\mathbf{C_1})$, w' is inserted in the tabu list with positive probability. Hence, we reach a state (w'', u') in $\mathcal{C}$ with $|u'| = |u| + 1$. Since, by definition, the length of the tabu list cannot decrease and since the tabu chain revisits (w', u) almost surely, we reached a contradiction.
- Second, assume that u is full. According to $(\mathbf{C_2})$, the last element of the tabu list is removed with positive probability. On one hand, (w', u) is visited infinitely often and we remove infinitely often the last element of the current tabu list. On the other hand, the walker never reaches the vertex w. Hence the number of occurrences of w in its tabu list decreases. Eventually, all occurrences of w in the tabu list are removed. Since (z, t) is visited infinitely often, the tabu list t cannot contain any occurrence of w. Once again, we reached a contradiction.

4 Optimal Update Rule for m-Free Graphs

For each positive integer m, we identify a non trivial class of graphs on which $FIFO_m$ gives the smallest mean hitting time, among all update rules of length at most m. Then, we describe a class of update rules that yield tabu random walks with the same law than $FIFO_m$ on this class of graphs.

Definition 3 **(m-Free Graphs).** *Let m be a positive integer. A graph is m-free if there does not exist any path $x_0, \ldots, x_k$ with length $k \geqslant 1$ that satisfies the four following conditions:*

1. *The vertex x_k has degree at least two.*
2. *For all integers j in $\{0, \ldots, k-1\}$, $x_j \neq x_k$.*
3. *All neighbors of x_k of degree at least two belong to $\{x_0, \ldots, x_{k-1}\}$.*
4. *$k + 2d \leqslant m$, where d is the number of neighbors of x_k of degree one.*

The idea behind Definition 3 is that for each m-free graph with $m \geq 2$, a walker who uses $FIFO_m$ always selects a destination that is not in his current tabu list. Hence, he avoids cycles of size less than or equal to m.

As direct consequences of the definition, note that *all* graphs are 1-free and that all $(m+1)$-free graphs are also m-free, for every positive integer m. Furthermore:

- A graph is 2-free if and only if it does not contain any triangle with one vertex of degree exactly two, namely, if and only if it does not possess any vertex x such that $V_x = \{y, z\}$ and $\{x, z\} \subseteq V_y$, where V_x and V_y are, respectively, the set of neighbors of x and y.
- Every $(m+1)$-*regular* graph, namely with all vertices of degree $m+1$, is m-free. Indeed, assume that $x_0, \ldots, x_k$ is a path that satisfies the four conditions stated above. Since $d = 0$, we infer that $k \leqslant m$. Yet, x_k has $m + 1 > k$ neighbors of degree at least two while the set $\{x_0, \ldots, x_{k-1}\}$ has k elements. Thus, the condition 3 is not satisfied and we reach a contradiction.
- Every graph with *girth* strictly greater than $m+1$, that is, where every cycle has at least $m+2$ edges, is m-free.

The following theorem states that for m-free graphs, and with no more than m memories, $FIFO_m$ is the optimal update rule.

Theorem 4. *Let m be a positive integer and R_k be an update rule of length $k \leqslant m$. On a m-free graph, for every two vertices x and y, $E_{(x,\varepsilon)}H_y(FIFO_m) \leqslant E_{(x,\varepsilon)}H_y(R_k)$.*

Note that being m-free is not a necessary condition to have, for every two vertices x and y, $E_{(x,\varepsilon)}H_y(FIFO_m) \leqslant E_{(x,\varepsilon)}H_y(R_k)$ (Theorem 4 only gives a sufficient condition). Indeed, consider the clique with vertex set $\{0, 1, 2\}$. The path $(x_0, x_1, x_2) = (0, 1, 2)$ ensures that the graph is not 2-free. Yet, for every two distinct vertices x and y and for every update rule R, $E_{(x,\varepsilon)}H_y(R) \geqslant 3/2$, while $E_{(x,\varepsilon)}H_y(FIFO_2) = 3/2$.

We now sketch a proof of Theorem 4.

Proof. Let m be a positive integer and let G be a m-free graph. Since G is m-free, we can show, by contradiction, that for every tabu random walk associated to $FIFO_m$, the walker does not visit a vertex if at least one occurrence of that vertex is in the current tabu list, except if the current vertex has degree one. Now, consider an ordered pair (x, y) of vertices of G and an update rule R_k of length k in $\{0, \ldots, m\}$. Let $(X_n(R_k))_{n=0}^{H_y(R_k)}$ be the associated tabu random walk on G that starts at x with empty tabu list and stops when it reaches y. Assume that i is an integer such that $X_i(R_k)$ has degree at least two while $X_{i+1}(R_k)$ has an occurrence in the tabu list $T_i(R_k)$. In particular, this implies that all neighbors of $X_i(R_k)$ have an occurrence in $T_i(R_k)$. We set $j = \min\{\ell \geqslant 2 : X_i(R_k) = X_{\max\{i-\ell,0\}}(R_k)\}$. We remove all steps of $(X_n(R_k))_{n=0}^{H_y(R_k)}$ from $i-j+1$ to i. By applying iteratively this operation, we obtain a random walk $(\widetilde{X}_n)_{n=0}^{\widetilde{H}_y}$ such that the walker does not visit a vertex if at least one occurrence of that vertex is in the current tabu list, except if the current vertex has degree one.

Applying the same scheme recursively, we can prove that $(\widetilde{X}_n)_{n=0}^{\widetilde{H}_y}$ follows the same law than the first $H_y(FIFO_m)$ steps of the tabu random walk $(X_n(FIFO_m))_{n=0}^{H_y(FIFO_m)}$ associated to the update rule $FIFO_m$, starting at (x, ε). Therefore, we infer that $E_{(x,\varepsilon)}\widetilde{H}_y = E_{(x,\varepsilon)}H_y(FIFO_m)$. By construction $\widetilde{H}_y \leqslant H_y(R_k)$ then we obtain $E_{(x,\varepsilon)}H_y(R_k) \leqslant E_{(x,\varepsilon)}H_y(FIFO_m)$.

From the above theorem, we can deduce that the non-backtracking random walk $(FIFO_1)$ is the fastest among all update rules of length less or equal to 1, since every graph is 1-free.

Corollary 5. *For every update rule R of length 0 or 1 and for every two vertices x and y of every graph, $E_{(x,\varepsilon)}H_y(FIFO_1) \leqslant E_{(x,\varepsilon)}H_y(R)$.*

Proposition 6 completes Corollary 5 and states that $FIFO_1$ is the only update rule of length 1 such that, on all graphs, all hitting times are smaller than those associated to the simple random walk, here represented by $FIFO_0$.

Proposition 6. *If R is an update rule of length 1 distinct from $FIFO_1$, then there exists a positive integer ℓ such that on the graph F_ℓ,[4] $E_{(1,\varepsilon)}H_0(FIFO_0) < E_{(1,\varepsilon)}H_0(R)$.*

Theorem 4 shows that $FIFO_m$ is the best update rule of length at most m for m-free graphs. In Theorem 7 below, we characterize a larger class of update rules which are optimal policies in that specific case; for example, LRU is in this class as stated by Corollary 8 (as a direct application of Theorem 7).

Theorem 7. *Consider a positive integer m and a m-free graph. Every update rule of length k in $\{0,\ldots,m\}$ yields tabu random walks with the same law as those associated to $FIFO_k$ if and only if it satisfies the two following conditions:*

- *If the tabu list is not full and does not contain the current vertex, then it is inserted.*
- *If the tabu list is full and does not contain the current vertex, then the last element is removed and the current vertex is inserted.*

Proof. The two conditions stated above imply that the walker never has any occurrence of its current position in its tabu list when the graph is m-free. Hence, the law of the tabu random walk is entirely determined by the update rule when the current vertex does not have any occurrence in the tabu list.

Conversely, if an update rule of length k differs from $FIFO_k$ when the current vertex does not have any occurrence in the tabu list, then the associated tabu random walks on a clique with $m+3$ vertices (which is a m-free graph) follow two distinct laws.

Corollary 8. *Let m be a positive integer. On every m-free graph, for every integer k in $\{0,\ldots,m\}$, the update rules LRU_k and $FIFO_k$ yield tabu random walks with same law.*

As a conclusion, within the class of m-free graphs and no more than m memories, we identified a class of optimal update rules which contains $FIFO_m$. For other cases, namely, for graphs that are not m-free or using more that m memories, the question is still open.

5 Impact of the Length of the Update Rules

We study the effect of the size of the memory of the walker, on the mean hitting times using 3 collections of update rules: $(FIFO_m)_{m\geqslant 0}$, $(LRU_m)_{m\geqslant 0}$, and $(RAND_m)_{m\geqslant 0}$. Our results shows that there is no general trend, *i.e.*, having more memory does not always increase the performances. To see this, we present comparisons based on four

[4] Flower graphs F_ℓ are defined in Section 5.

particular collections of graphs: the cliques $(K_r)_{r\geqslant 2}$, the lollipops $(L_r)_{r\geqslant 3}$, the lines $(P_r)_{\geqslant 2}$, and the flowers $(F_\ell)_{\ell\geqslant 1}$. For a given update rule, R_m of length m, we study the mean hitting time $h(R_m) = E_{(1,\varepsilon)}H_0(R_m)$ of the vertex 0 for a walker that starts from vertex 1 with an empty tabu list. The next paragraphs present the graphs and the comparisons. All the results are summed up in Proposition 9.

m-Free Graphs. Within the class of m-free graphs and with a $FIFO$ update rule of length less than m, the greater the length is, the more efficient the algorithm is. Precisely, for every positive integer m, and for all integers k such that $k \leq l \leq m$, we have $h(FIFO_l) \leq h(FIFO_k)$ on every m-free graph. This is a direct application of Theorem 4.

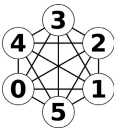


Fig. 2. The clique K_6

Cliques. For every integer $r \geq 2$, the clique K_r is the complete graph with vertex set $\{0, \ldots, r-1\}$: each vertex is neighbor of all other vertices. As an example, the clique K_6 is drawn in Figure 2. Note also that K_r, for $r > 2$ is $(r-2)$-free.

In our technical report [9], we compute on the clique K_r an *analytic expression* of $h(R_m)$ for a family of update rules that contains $(FIFO_m)_{m\geqslant 0}$, $(LRU_m)_{m\geqslant 0}$, and $(RAND_m)_{m\geqslant 0}$. Using this expression, we compare all these update rules and conclude that the larger the length of the update rule is, the smaller the mean hitting times are.

Lollipops. For every integer $r \geqslant 3$, L_r denotes the lollipop graph with vertex set $\{0, \ldots, r-1\}$ such that the vertices $2, \ldots, r-1$ form a clique with $r-1$ elements and the vertex 1 has 0 and 2 as neighbors. As an example, the lollipop L_6 is drawn in Figure 3.

We use lollipop graphs to compare:

1. $RAND_3$ against $RAND_2$;
2. $RAND_k$ with $k \geq 4$ against $RAND_m$ with $1 \leq m < k$; and
3. LRU_k with $k \geq 3$ against LRU_m with $1 \leq m < k$.

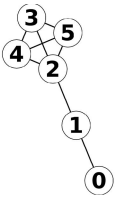


Fig. 3. The lollipop L_6

A walker on the lollipop L_r that starts at the vertex 1 and does not hit the vertex 0 at its first move, must stay on the set of vertices $\{3, \ldots, r-1\}$ until its tabu list is full. Thus, increasing the length of the update rule may raise the mean hitting time and this is actually the case for the comparisons above.

Lines. For every integer $r \geqslant 2$, P_r denote the graph line with vertex set $\{0, \ldots, r-1\}$ and edge set $\{\{i, i+1\},\ i \in \{0, \ldots r-2\}\}$. As an example, the graph P_4 is drawn in Figure 4.

Line graphs are used to compare:

1. $FIFO_k$ with $k \geq 3$ against $FIFO_m$ with $0 \leq m < k$;
2. LRU_k with $k \geq 3$ against LRU_0 (the simple random walk);
3. $RAND_k$, with $k \geq 4$ against $RAND_0$ (the simple random walk); and
4. $RAND_3$ against $RAND_m$ with $m = 0, 1$.

Fig. 4. The line P_4

In the above comparison, the length of the update rule raises the mean hitting time on a line. Indeed, assume that m is a positive integer and consider a walker on P_r that starts at the vertex 1 and does not hit the vertex 0 at its first move. First, the walker must go to the end of the line, that is to say the vertex $r-1$, without backtracking. Then, its tabu list is almost surely $(r-2)\cdots(r-m-1)$. Thus, the walker performs a simple random walk, until it reaches a vertex with a neighbor not included in the tabu list. The duration of the simple random walk behavior increases with the length of the update rule. Next, the walker goes to the vertex 0 without backtracking.

Flowers. For all positive integers ℓ, we define the graph F_ℓ with vertex set $\{0,\ldots,2\ell+1\}$ as follows. Initially, the vertices 0 and 1 are isolated and for each integer x in $\{1,\ldots,\ell\}$, the vertices $2x$ and $2x+1$ are neighbor. Then, each vertex is linked to the vertex 1, except the vertex 1 itself. As an example, the flower F_3 is drawn in Figure 5.

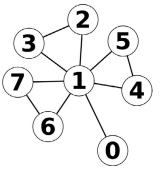


Fig. 5. The flower F_3

We deal with the flower graphs to compare:

1. $FIFO_2$ against $FIFO_k$, $k=0,1$;
2. LRU_2 against LRU_k, $k=0,1$; and
3. $RAND_2$ against $RAND_k$, $k=0,1$.

For the above comparison, increasing the length of the update rule raises the mean hitting time in a flower graph. Indeed, consider a walker in F_ℓ that starts at the vertex 1 and does not hit the vertex 0. Without loss of generality, assume that he reaches vertex 2 at its first move. Now, the mean return time to the vertex 1 increases with the length of the update rule. After having returned to the vertex 1, the walker either hits 0 or reaches again the previous situation and must return again to vertex 1.

Results. The next proposition summarizes the above results and presents a complete view of the impact of the length on the update rules for $(FIFO_m)_{m\geqslant 0}$, $(LRU_m)_{m\geqslant 0}$ and $(RAND_m)_{m\geqslant 0}$.

Proposition 9. *On the graph written at k-th row and m-th column,*

1. $h(FIFO_k) > h(FIFO_m)$*:*

k \ m	0	1	2	$m \geqslant 3$
0	$\emptyset$	K_3	K_3	K_3
1	$\emptyset$	$\emptyset$	K_3	K_3
2	F_7	F_4	$\emptyset$	K_4
$k \geqslant 3$	P_4	P_4	P_4	$\begin{cases} P_{m+2} & \text{if } m < k, \\ \emptyset & \text{if } m = k, \\ K_{k+2} & \text{if } m > k. \end{cases}$

2. $h(LRU_k) > h(LRU_m)$:

k \ m	0	1	2	$\geqslant 3$
0	$\emptyset$	K_3	K_3	K_3
1	$\emptyset$	$\emptyset$	K_3	K_3
2	F_7	F_4	$\emptyset$	K_4
$\geqslant 3$	P_4	L_4	L_4	$\begin{cases} K_{k+2} & \text{if } m > k\,, \\ \emptyset & \text{if } m = k\,, \\ L_{m+2} & \text{if } m < k\,. \end{cases}$

3. $h(RAND_k) > h(RAND_m)$:

k \ m	0	1	2	3	$m \geqslant 4$
0	$\emptyset$	K_3	K_3	K_3	K_3
1	F_5	$\emptyset$	K_3	K_3	K_3
2	F_5	F_6	$\emptyset$	K_4	K_4
3	P_4	P_4	L_5	$\emptyset$	K_5
$k \geqslant 4$	P_4	L_5	L_5	L_5	$\begin{cases} K_{k+2} & \text{if } m > k\,, \\ \emptyset & \text{if } m = k\,, \\ L_{m+2} & \text{if } m < k\,. \end{cases}$

In each table, the symbol $\emptyset$ means that no such graph exists. Actually, symbol $\emptyset$ appears in two disjoint cases : when $k = m$ or when we compare $FIFO_0$ to $FIFO_1$. (In this latter case, for all graphs and all vertices x and y, $E_{(x,\varepsilon)}H_y(FIFO_0) \geqslant E_{(x,\varepsilon)}H_y(FIFO_1)$, by Corollary 5.)

The above proposition shows that, for the three studied collections of update rules, there is no general trend: increasing the length of the memory does not always lead to a gain of performance and may even be a penalty in some cases.

6 Conclusion and Perspectives

We analyzed classes of tabu random walks characterized by their update rules. Our goal was to study the impact of the choice of an update rule on the performance of tabu random walks. We focus on classes of update rules, for which we give a necessary and sufficient condition that ensures the finiteness of the mean hitting time of the associated tabu random walk on every graph. Then, we exhibit non-trivial classes of graphs, namely the m-free graphs, on which we exhibit the optimal update rules among those of length at most m. Finally, we study the impact of the tabu list length on the efficiency of the walk. This latter study shows that, except in one case (namely $FIFO_1$), modifying the length of the tabu list does not guarantee a better hitting time in all cases.

Our results could be extended to a larger class of update rules for which more removals are allowed; for example, the list could be reset regularly. A preliminary study

shows that this extension should be carefully done: we believe that the necessary and sufficient condition of Theorem 2 could be adapted and that the result on m-free graphs still holds. In future works, we would also like to compare the relative performance of different update rules of same length. As a first step, we know that given a positive integer m, $FIFO_m$ is faster than LRU_m on line graphs, and LRU_m is faster than $RAND_m$ on lollipop graphs (see [9]).

References

1. Ochirkhand, E.O., Minier, M., Valois, F., Kountouris, A.: Resilient networking in wireless sensor networks. Research report, Inria (2010)
2. Ponsonnet, C.: Secure probabilistic routing in wireless sensor networks. Master thesis, Laboratoire Verimag (2011)
3. Lovász, L.: Random walks on graphs: A survey. In: Miklós, D., Sós, V.T., Szönyi, T. (eds.) Combinatorics, Paul Erdös is Eighty. Bolyai Society Mathematical Studies, vol. 2, pp. 1–46. János Bolyai Mathematical Society (1993)
4. Aldous, D., Fill, J.: Reversible Markov Chains and Random Walks on Graphs. Book in preparation (20XX), http://www.stat.berkeley.edu/~aldous/RWG/book.html
5. Slade, G.: The self-avoiding walk: A brief survey. In: Blath, J., Imkeller, P., Roelly, S. (eds.) To appear in Surveys in Stochastic Processes, Proceedings of the Thirty-third SPA Conference in Berlin, 2009. The EMS Series of Congress Reports (2010)
6. Li, K.: Performance analysis and evaluation of random walk algorithms on wireless networks. In: IPDPS Workshops, pp. 1–8 (2010)
7. Altisen, K., Devismes, S., Lafourcade, P., Ponsonnet, C.: Routage par marche aléatoire à listes tabous. In: Algotel 2011, Cap Estérel, Mai 23-26 (2011)
8. Ortner, R., Woess, W.: Non-backtracking random walks and cogrowth of graphs. Canad. J. Math. 59(4), 828–844 (2007)
9. Altisen, K., Devismes, S., Gerbaud, A., Lafourcade, P.: Comparisons of mean hitting times for tabu random walks on finite graphs. Technical report, Laboratoire Verimag (2011), http://www-verimag.imag.fr/TR/TR-2012-6.pdf
10. Hughes, B.: Random walks and random environments, vol. 1. Oxford University Press (1995)
11. Levin, D., Peres, Y., Wilmer, E.: Markov chains and mixing times. American Mathematical Society (2009)

Online Graph Exploration with Advice*

Stefan Dobrev[1], Rastislav Královič[2], and Euripides Markou[3]

[1] Institute of Mathematics, Slovak Academy of Sciences, Bratislava, Slovakia
Stefan.Dobrev@savba.sk
[2] Department of Computer Science, Comenius University, Bratislava, Slovakia
kralovic@dcs.fmph.uniba.sk
[3] Department of Computer Science & Biomedical Informatics,
University of Central Greece, Lamia, Greece
emarkou@ucg.gr

Abstract. We study the problem of exploring an unknown undirected graph with non-negative edge weights. Starting at a distinguished initial vertex s, an agent must visit every vertex of the graph and return to s. Upon visiting a node, the agent learns all incident edges, their weights and endpoints. The goal is to find a tour with minimal cost of traversed edges. This variant of the exploration problem has been introduced by Kalyanasundaram and Pruhs in [18] and is known as a *fixed graph scenario*. There have been recent advances by Megow, Mehlhorn, and Schweitzer ([19]), however the main question whether there exists a deterministic algorithm with constant competitive ratio (w.r.t. to offline algorithm knowing the graph) working on all graphs and with arbitrary edge weights remains open. In this paper we study this problem in the context of *advice complexity*, investigating the tradeoff between the amount of advice available to the deterministic agent, and the quality of the solution. We show that $\Omega(n \log n)$ bits of advice are necessary to achieve a competitive ratio of 1 (w.r.t. an optimal algorithm knowing the graph topology). Furthermore, we give a deterministic algorithm which uses $O(n)$ bits of advice and achieves a constant competitive ratio on any graph with arbitrary weights. Finally, going back to the original problem, we prove a lower bound of $5/2 - \epsilon$ for deterministic algorithms working with no advice, improving the best previous lower bound of $2 - \epsilon$ of Miyazaki, Morimoto, and Okabe from [20]. In this case, significantly more elaborate technique was needed to achieve the result.

1 Introduction

The exploration of an unknown environment is a well studied problem under many different scenarios. This problem appears in many areas, such as terrain exploration by robots, network exploration by agents, maintaining security of large networks or searching for data in the internet and ad-hoc networks. As

* E. Markou was supported in part by a research grant offered by the Ministries of Education of Greece and Slovakia (Bilateral Exchange Programme for Researchers) and by THALES: ALGONOW project funded by the Greek Ministry of Education.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 267–278, 2012.

the agent has initially limited knowledge about the network topology and this knowledge grows only by the agent observing its immediate neighbourhood, in order to perform exploration (and/or construct a complete map of the network) the agent has to visit each node.

In the online graph exploration problem an agent starts at a node s of an undirected labeled graph $G = (V, E)$, with $|V|$ denoted by n. Each edge $e \in E$ has a non-negative weight, also called length or cost of e. The agent has no knowledge about the topology of G. The task of the agent is to visit every node of the graph and return to s. The agent can move only along the edges of G, each time paying the respective edge cost. In the particular variant we consider in this paper, when the agent arrives at a node $u \in G$, it learns all incident edges, their weights and their endpoints. This scenario has been introduced by Kalyanasundaram and Pruhs in [18] and is known as a *fixed graph scenario*. While learning the endpoints of the incident edges is stronger than the typical exploration scenario, it does have justification (see [18] and [19]); it also corresponds to previously studied *neighbourhood sense of direction* [8].

The quality of an exploration algorithm under the above scenario is usually measured by a *competitive analysis* ([4]), which compares the solution of an algorithm with an optimal offline solution, i.e., the solution of an optimal algorithm which has access to a complete and accurate map of the network. This analysis is complicated by the fact that the underlying offline problem corresponds to the *Traveling Salesman Problem* (TSP), which is known to be NP-hard, even to get a constant-approximation (e.g., see [14]).

Related Work. A simple and fast heuristic for the traditional TSP offline setting which has been studied a lot is the greedy algorithm *Nearest Neighbor* (NN): Once at a node u, go to the closest yet unexplored vertex v and repeat the process until all vertices have been explored. This algorithm also applies in the online setting, achieving competitive ratio of $\Theta(\log n)$ ([21]), which is tight even on planar unit-weight graphs ([16]).

While NN is non-competitive on general graphs, it performs quite well (with competitive ratio of 3/2 ([1])) on simple cycles. A close lower bound of 5/4 was also proved in [1]. These results for cycles have been later improved to $\frac{1+\sqrt{3}}{2}$ matching lower and upper bound [20].

For graphs in which all edges have the same weight, a *Depth First Search* (DFS) is 2-competitive, as the weight of a Minimum Spanning Tree (MST) is a lower bound. This has been shown to be optimal in [20]. A sophisticated generalization of DFS (named `ShortCut`), introducing a parameterized condition which determines when to diverge from DFS, has been proposed in [18]. `ShortCut` has been shown to achieve competitive ratio of 16 in planar graphs; it has been long-standing hypothesis that it is in fact constant competitive. `ShortCut` has been reformulated in [19] and the upper bound has been generalized to $16(1 + 2g)$ for graphs of genus at most g. However, it has been shown in [19] that neither of these algorithms is constant competitive in general graphs with arbitrary weight. In fact there are classes of graphs for which their competitive ratio is arbitrarily large. Finally, a generalization of DFS that can be seen as a hierarchical DFS

was shown to be constant competitive on graphs with a bounded number of different edge weights. A slight generalization of this algorithm achieves $\Theta(\log n)$ competitive ratio for graphs with arbitrary weights.

Advice Complexity. The impact of additional (typically structural) information on complexity of algorithms has been a longstanding and rich field of study. Impact of various aspects of structural information (e.g. knowledge of network size and/or network topology, presence of *sense of direction*, availability of distinct node IDs) has been extensively studied for various problems. In general, this information has been of qualitative type, i.e. the questions asked were of the type "what is the impact of presence/absence of specific structural information?".

A new line of research focusing on quantitative aspects of such information has has recently become popular. The idea is to provide the algorithm/agent with some additional information (*advice*) given explicitly as a binary string, thus allowing to measure the information quantitatively. We use the model from [5,10] where the advice is given to the agent at the beginning of the algorithm. Alternatively, the advice could be stored in a distributed fashion in the nodes of the network (see e.g. [9,11,12,13,17]), and the maximum or average size of the advice per node be considered. The advice encodes problem-relevant information about unknown facts (i.e. topology in case of distributed algorithms) and can be seen as computed by an oracle (of unlimited power) that knows the missing information the algorithm wants to use. This approach allows to precisely measure the amount of additional information provided to the agent and facilitates study of the tradeoff between the size of the advice, and the quality of the solution.

In the context of online algorithms, analogous concept has been independently proposed in [6], and has been developed in two ways: the model from [7] considers that the algorithm receives, with each request, some fixed amount of b bits of advice. In the model from [3] (see also [2,15]), on the other hand, the whole advice is given to the algorithm at the beginning.

Our Results. Our primary interest is in the study of the tradeoff between the advice size and the quality of the solution for the case of general graphs with arbitrary weights. As a first step, we show (in Section 2.1) that in order to have an optimal algorithm with competitive ratio strictly 1, advice of size $\Omega(n \log n)$ bits is needed.

The primary question we are interested in is what is the smallest advice with which there is a constant competitive algorithm. This can be seen as a relaxation of the original question whether there is a constant competitive algorithm with no advice at all. We provide an upper bound (in Section 3), presenting an algorithm that achieves constant competitive ratio $6 + \epsilon$ using $O(n)$ bits of advice. This result seems rather weak as we had to pay $O(n)$ bits of advice to reduce the competitive ratio by a factor of $O(\log n)$. However, it took a quite elaborate algorithm to achieve even this result. As the algorithm has to make many decisions over the whole network, using $o(n)$ bits of advice and still achieving constant competitive ratio would require significant new insight.

Going back to the original problem with no advice, we prove (in Section 2.2) that without advice the competitive ratio of any deterministic online algorithm cannot be less than $5/2 - \epsilon$. The best previous lower bound of $2 - \epsilon$ is from [20] and concerns graphs with unit weights (for which there is also a matching upper bound). Our lower bound is significantly more involved, necessarily employing edges of many different weights in an elaborate hierarchical structure. Due to space constraints the proofs of lemmas and theorems and some of the more technical parts will appear in the full version of the paper.

2 Lower Bounds

2.1 Advice Size for Optimal Solution

Let $\{v_0, v_1, \ldots, v_{n-1}\}$ be a set of vertices. Define $w(v_i) = n - i$. Denote by K_n^w the clique on vertices $v_0, v_1, \ldots, v_{n-1}$ in which the edge between vertices v_i and v_j is of weight $w(v_i, v_j) = \max(w(v_i), w(v_j))$.

Lemma 1. *There is a unique (up to reversal) walk π (visiting all nodes of K_n^w) of a minimum cost in K_n^w with endpoints v_0, v_{n-1}. Furthermore, $\pi = \{v_0, v_1, \ldots, v_{n-1}\}$.*

Consider now K_n^w in which the adversary assigns the IDs visible to the exploring agent. Hence, when at node v_i, the agent can from the weights of the incident edges deduce i and the IDs of the nodes v_j for $j < i$. However it cannot distinguish between the $n - i$ edges of weight $w(i)$ leading to not-yet-visited vertices. Therefore, in order to ensure the vertices are visited in the optimal order $v_0, v_1, \ldots, v_{n-1}$, the agent needs advice of size $\log(n - i)$ at vertex i. Assuming the agent starts at vertex v_0 and summing up over all vertices yields $\Omega(n \log n)$ bound on the advice.

However, since the reverse of π is also an optimal path, the adversary can give the advice (of size $\log n$) which edge leads to v_{n-1}. Once the agent is in v_{n-1}, it can from the weight of the incident edges deduce the remainder of the optimal traversal sequence. Hence an agent could complete a cycle visiting all nodes of K_n^w using just $\log n$ bits of advice. In order to prevent this exploit, consider the graph G consisting of two copies of K_n^w and two additional edges of unit weight. An example with two copies of K_8^w is shown in Figure 1. Using Lemma 1 we show that the optimal cycle including all nodes of G is:

$$\{v_0, v_1, \ldots, v_{n-1}, v'_{n-1}, v'_{n-2}, \ldots, v'_0, v_0\}$$

Hence, to traverse this cycle, the agent needs $\Omega(n \log n)$ bits of advice in at least one copy of K_n^w.

Theorem 1. *There is a family of graphs for which any optimal-cost algorithm solving the graph exploration problem needs $\Omega(n \log n)$ bits of advice.*

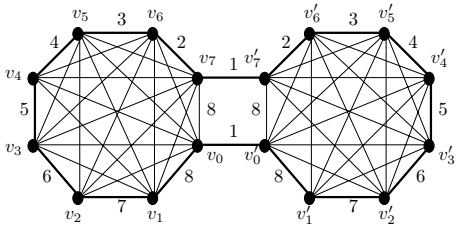


Fig. 1. The lower bound graph. The bold line is the optimal exploration path.

2.2 Lower Bound for the Case of No Advice

Theorem 2. *For any deterministic algorithm $\mathcal{A}$ and any $\frac{1}{2} > \varepsilon > 0$ there exists a graph G_{lb} such that the competitive ratio of $\mathcal{A}$ is at least $\frac{5}{2} - \varepsilon$. Moreover, G_{lb} has $r^{O(\log r)}$ vertices, where $r = 2/\varepsilon$.*

The lower bound graph G_{lb} consists of a *backbone* cycle and additional *return* and *skip* edges. At the highest level, G_{lb} includes x level-k (x and k will be determined later, x is odd) blocks connected in a backbone cycle, and $x - 1$ additional skip edges to be described later. Each block of level-i, where $i > 1$ consists of x sub-blocks of level-$(i-1)$ forming a line connected by level-$(i-1)$ backbone edges. The agent starts exploration in the middle of a level-k block. Each block has a right and left side; which side is which is decided based on algorithm's actions. In particular, the side whose endpoint is first reached by the agent is by definition the right side; an adversary decides how the left and right sides of neighbouring blocks align.

Let v be the rightmost vertex of a level-i block B such that B is the highest level block for which v is the rightmost vertex. Then v is connected to the next level-i block by two level-i edges: the backbone edge leading to the leftmost vertex of the next level-i block, and the skip edge leading to the middle of the next level-i block. There are two exceptions (the middle block has two skip edges, and one block has no skip edges). Additionally, there is a level-i return edge from v to the middle of the leftmost sub-block of B. The weights of these three level-i edges incident to v are the same and are equal to the cost of the minimum cost (we also call it shortest) path (by the cost of a path we mean the sum of the weights of the edges of the path) connecting the endpoints of the return edge and not using those three level-i edges. Note that if v is also the rightmost vertex of a lower-level block B' then B' does not contain a return edge of its level. When given a choice (i.e., the algorithm wants to traverse an edge whose endpoint's ID has not yet been seen) the adversary's order of preference is return edge, then skip edge, then backbone edge.

Level-1 block is analogous: It consists of x vertices connected in a line by backbone edges of weight 1. There is a difference, though: The return edge leads

to the second leftmost vertex[1]. As a consequence, the right side of a level-1 block is the side in which the third vertex from the end is visited first[2]. The structure of a level-i block is captured in Figure 2.

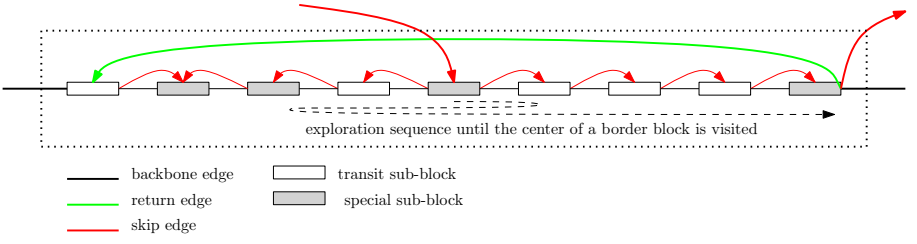


Fig. 2. The structure of a level i block (this one is transit). Fat edges are of level i.

In the following, by the cost of a path we mean the sum of the weights of the edges of the path. Let r_i denote the cost of the shortest path from the middle of a level-i block to its rightmost/leftmost vertex (note that these distances are the same) Let t_i denote the cost of the shortest path using edges of level at most $i-1$ and connecting the endpoints of a level-i return edge. Let o_i denote the cost of the backbone path from the left-most vertex to the right-most vertex of a level-i block.

Let us classify the blocks as *transit* and *special*. A block B of level i is *transit*, if and only if all of the following conditions hold:

- B is entered for the first time via the skip edge leading into its center
- at that time, exactly one neighbouring block of B has not been visited – the block B'
- B' is visited for the first time via a skip edge from B
- B contains a return edge of level i

All other blocks are *special*. Observe that in an optimal algorithm, a transit block has at most 4 special sub-blocks, while a special block may have up to 5 special sub-blocks.

Let us denote by e_i and $\tilde{e}_i$ the minimal cost (perhaps over several visits) incurred by the agent in a level-i transit and special block, respectively, until all vertices of the block have been visited. For the transit blocks, we charge to the block also the cost of arriving to and leaving the block (if there is such activity) in the time period between the first arrival of the agent to B and the first leaving of the agent towards B'. From the definition of G_{lb} we have

[1] The reason is that the ID of the leftmost vertex might be already known to the algorithm, allowing to distinguish the incident level-i edges at the rightmost vertex.

[2] Once this vertex is visited, it knows the ID of the second-from-end vertex which would allow to recognize the endpoint of the return edge.

$$
\begin{array}{ll}
r_1 = \frac{1}{2}(x-1) & r_{i+1} = \frac{1}{2}(x+1)r_i + \frac{1}{2}(x-1)t_i \\
t_1 = x-2 & t_{i+1} = (x-1)t_i + xr_i \\
o_1 = x-1 & o_{i+1} = xo_i + (x-1)t_i \\
e_1 \geq \frac{5}{2}(x-1) & e_{i+1} \geq (x-4)e_i + 4\tilde{e}_i + (x-1)t_i + t_{i+1} + r_{i+1} - r_i = \\
 & \quad (x-4)e_i + 4\tilde{e}_i + \frac{5}{2}(x-1)t_i + \frac{3x-1}{2}r_i \\
\tilde{e}_1 \geq x-1 & \tilde{e}_{i+1} \geq (x-5)e_i + 5\tilde{e}_i + (x-1)t_i
\end{array}
$$

Consider the highest-level ring of blocks of level k. As there are two special blocks in an optimal algorithm (the starting one, and its left neighbour), the exploration cost is at least $(x-2)e_k + 2\tilde{e}_k + xt_k$. The optimal traversal cost (of an algorithm using a map of the graph) is $xo_k + xt_k$. Putting this in ratio yields the approximation factor

$$\frac{(x-2)e_k + 2\tilde{e}_k + xt_k}{xo_k + xt_k} = \frac{5}{2}$$

Corollary 1. *The competitive ratio of any deterministic algorithm on n-vertex graphs is at least $\frac{5}{2} - 2^{-O(\sqrt{\log n})}$.*

3 Upper Bounds

Let M be the weight of a Minimum Spanning Tree (MST) of G. We design the exploration algorithm to incur cost $O(M)$ and hence achieve constant approximation ratio. Intuitively the algorithm classifies edges of G into groups depending on their weight and leads (by providing $O(n)-$ bits of advice) the agent to explore G by traversing components of an MST and some not very 'heavy-weight' edges connecting those components. All the advice described is stored in self-delimited way; this ensures that the cost of an advice of number s is $O(\log s)$ regardless of the potential range the value s is from. As a first step, the advice given is n and $l = \lceil \log(M/n) \rceil$. Although l can be unbounded w.r.t. n, it can be encoded in $O(\log n)$ bits in the following way: Advice (n', p, l') is given, interpreted as follows: Keep exploring the cheapest outgoing edge from the currently explored subgraph until n'-th vertex is encountered. Consider its p-th incident edge e, let $w(e)$ be its weight. Then $l = \lceil \log w(e) \rceil + l'$. (n', p, l') are chosen in such way that e is the first encountered edge of weight between M/n^2 and M. Note that such an edge must exists, otherwise the MST weight would not be M. Observe that $O(\log n)$ bits are sufficient to encode (n', p, l') and the total exploration cost until e is found is $O(M)$: a)$l' \leq \log n$, and b)the cost of reaching the cheapest outgoing edge is $O(M/n)$ since each so-far explored edge is of weight at most $O(M/n^2)$, and this has to be repeated for at most n times until a heavier edge is found.

Define for each edge e its level $l(e)$ as follows: If $\log(w(e)) < l$, then $l(e) = 0$, otherwise $l(e) = \lceil \log(w(e)) \rceil - l$. Note that in the MST there at most $n/2^i$ edges of level i. Define G_i to be the graph induced in G by the edges of level at most i. Denote by $G_i(v)$ the connected component of G_i containing vertex v.

From a high level view, the algorithm tries to mimic the MST of G, with the following modifications: a) The G_0 components are explored using DFS, as the total overhead in them w.r.t. to the MST is $O(M)$, and b) for each level-i MST edge connecting two G_0 components, the algorithm is able to identify (and traverse) a level i edge (let us call it a *tree edge*) connecting them, incurring cost of at most 6 level-i edge traversals (leading to $O(M)$ overall cost).

The main problem is that the tree edges cannot be encoded explicitly, as that might cost $O(n \log n)$ advice bits. Note that all special nodes that need advice to be stored in them can be encoded in $O(n)$ bits, by storing in each special node the number of newly visited nodes to skip until the next special node is encountered. Hence, it is sufficient to focus on how to efficiently (in terms of advice size and incurred traversal cost) identify the tree edges incident to a source vertex v. As there are at most $n/2^i$ tree edges of level i, we can afford to spend $O(i)$ bits per tree edge of level i; $O(\log i)$ bits are in fact sufficient for our algorithm.

Let us call an edge *unexplored* if one of its endpoints has not yet been visited by the agent. A level-i edge (u, v) is an out_i edge if $v \notin G_{i-1}(u)$, otherwise it is an in_i edge. In the easiest case, all level-i unexplored edges incident to v are out_i edges. This is indicated by advice $\{Out, i, 0\}$ at v. In such case, the algorithm can safely cross the incident level-i unexplored edges and recursively explore the corresponding components. However, it might be the case that there are in_i edges incident to v. In order to avoid taking them (and paying unnecessarily high cost), a $\{Wait, i, 0\}$ advice is given. In such case, the algorithm ignores for now the level-i edges incident to v, with the promise that it will return later when only out_i edges remain unexplored. The right moment to return to v is when the last in_i edge (v, w) incident to v becomes explored (i.e., when the agent arrives to w). In such case, we say that w is the trigger vertex at level-i for v. In fact, w can be a trigger vertex for several vertices. This is indicated by an advice tuple $\{Trigger, i, mult\}$ at w, where $mult$ is the number of vertices for which w is trigger. It is too costly to store explicitly for which of w's neighbours it is the trigger. Instead, w checks how many of its neighbours are waiting for trigger. If that number is equal to $mult$, then w knows that it can trigger all its level-i neighbours that are waiting for the trigger. However, it may be the case that w has more level-i neighbours that are waiting for level-i trigger. In such case, the exploration proceeds without triggering, with the promise that once w learns whom to trigger (we call it that w is *released*), it will do so. A vertex v being triggered (at level i) by w means that the agent travels from w to v, explores the level-i edges incident to v and returns to w. Before the return to w, the agent notifies[3] all level-i triggers incident to v that v is not waiting anymore for a trigger. This might release some triggers. In such case, the agent will visit the released triggers from v before returning to w. In fact, such triggering and releasing can cascade several levels. Nevertheless, we will show that the overall cost is still $O(M)$. The pseudocode of the algorithm is given in Algorithm 1 and Algorithm 2.

[3] Note that this is just an internal calculation in the agent's data structures, no actual traversal is needed.

Algorithm 1. Exploration with linear advice

```
1: procedure ExploreWithAdvice
2:     Initialize n, M and l as described above, finishing at vertex v
3:     Let next, Waiting(v, lvl) and Trigger(v, lvl) be global variables
4:     next ← ReadAdvice(integer)
5:     call Expand(v)
6:     return to the starting vertex
7: end procedure
1: procedure Expand(v)
2:     next ← next − 1
3:     if next = 0 then
4:         next ←ReadAdvice(integer)
5:         adviceList ←ReadAdvice(list of triples of integers)
6:         call ExpandNeighbours(v, 0)
7:         for every tuple {type, lvl, mult} in adviceList do
8:             if type = Out then
9:                 call ExpandNeighbours(v, lvl)
10:            else if type = Trigger then
11:                W ← {u : l((v, u)) = lvl ∧ Waiting(u, lvl) = True}
12:                if mult = |W| then
13:                    call TriggerNeighbours(v, lvl, W)
14:                else
15:                    Trigger(v, lvl) ← mult
16:                end if
17:            else                                          ▷ type = Wait
18:                Waiting(v, lvl) ← True
19:            end if
20:        end for
21:    else
22:        call ExpandNeighbours(v, 0)
23:    end if
24: end procedure
```

In order to complete the description, the advice given (i.e., which vertices are special, and for which levels) has to be specified. Unfortunately, it is not possible to reflect in a straightforward manner the structure of the MST edges connecting the G_0 components: Consider the scenario shown in Figure 3. In this case, both $G_{i-1}(w)$ and $G_{i-1}(w')$ will be reached from v, although (u, w) might be the MST edge. The solution is simple: drop u as a special vertex, the cost of reaching w from v is at most twice the cost of reaching it from u. More generally, the special vertices can be computed as follows:

Simulate the run of the exploration algorithm and whenever you come to a vertex v with out_i edges, mark it as special for level i and add the corresponding tuple to the advice. Note that this may create many connections to the same G_0 component C, potentially substantially increasing advice size. However, these connections will all stop to be out_i edges when C is visited and fully explored (note that the agent returns from a component only after it has been fully

Algorithm 2. Exploration with linear advice – helper procedures

```
 1: procedure ExpandNeighbours(v, lvl)
 2:     for all incident edges (v, u) of level lvl do
 3:         if u has not yet been expanded then
 4:             go to u
 5:             call Expand(u)
 6:             return to v
 7:         end if
 8:     end for
 9:     if lvl > 0 then
10:         Waiting(v, lvl) ← False
11:         T ← {u : l((u, v)) = lvl ∧ Trigger(u, lvl) > 0}
12:         for all u ∈ T do
13:             Trigger(u, lvl) ← Trigger(u, lvl) − 1
14:             W ← {w : l((u, w)) = lvl ∧ Waiting(u, lvl) = True}
15:             if Trigger(u, lvl) = |W| then
16:                 go to u
17:                 call TriggerNeighbours(u, lvl, W)
18:                 return to v
19:             end if
20:         end for
21:     end if
22: end procedure
 1: procedure TriggerNeighbours(v, lvl, W)
 2:     for all u ∈ W do
 3:         go to u
 4:         call ExpandNeighbours(u, lvl)
 5:         return to v
 6:     end for
 7:     Trigger(v, lvl) ← 0
 8: end procedure
```

explored). If it happens that all out_i edges of a special vertex v stop being out_i before v had a chance to explore them, the tuple for v, i (and possibly the corresponding Trigger tuple) are removed from the advice. This leaves at most $n-1$ Out/Wait tuples (plus corresponding Trigger tuples) in the advice. Let us classify the edges traversed by the algorithm as follows: i) *traverse edges*: the edges traversed on lines 4 and 6 of ExpandNeighbours, ii) *trigger edges*: the edges traversed on lines 3 and 5 of TriggerNeighbours, and iii) *release edges*: the edges traversed on lines 16 and 18 of ExpandNeighbours. Note that no other edges are traversed by the algorithm.

Lemma 2. *The traverse edges form a spanning tree of G of weight $O(M)$, while the total advice used by the algorithm is of size $O(n)$ bits.*

Note that each release edge can be charged to a corresponding trigger edge, which itself can be charged to the corresponding traversal edge (all of them of the same level). Combined with Lemma 2 this yields:

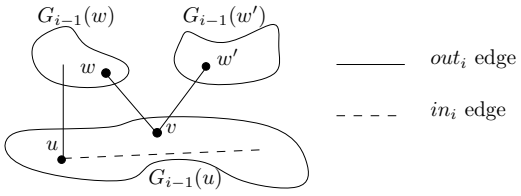


Fig. 3. u is visited before v, but v is the first one to expand its neighbours

Theorem 3. *Algorithm 1 explores an unknown n-node graph using advice of size $O(n)$ and incurring cost linear in the optimal exploration cost.*

A more careful analysis shows that exploration cost is at most $3W$ times the weight of the MST, where W is the ratio between the weights of the cheapest and the costliest edge of a level. Combined with the fast that the weight of the MST itself is a 2-approximation of the optimal cost, this yields approximation ratio of $6W$. In our case, we have chosen $W = 2$ for simplicity; W can be reduced to $1 + \epsilon$ by choosing narrower levels, at the expense of increasing the advice size (by increasing the number of levels). However, for any constant ϵ, the resulting advice size is still linear, providing a $6 + \epsilon'$ approximation bound with linear advice.

4 Conclusion

The question of whether there exists a constant competitive deterministic algorithm for the exploration problem on general graphs with arbitrary weights remains open. Hence adding to an algorithm the capability of accessing an advice seems a natural step for getting positive results. This is the case especially if it turns out that the answer to the above question is negative.

Our original aim was to come up with an algorithm using a small (polylogarithmic) advice and achieving constant competitive ratio. The hope was that such algorithm can perhaps be adapted to not need the advice at all. However, we did not succeed in this task and were only able to provide an algorithm using $O(n)$ bits of advice.

The principal problem lies in the fact that the algorithm has to make many decisions over the whole network. Hence, even reducing the advice to $o(n)$ requires new insight and would be a significant progress. From the lower bound side, we were able to raise the lower bound from $2-\epsilon$ to $5/2-\epsilon$, although breaking the barrier of 2 required use of many different edge weights and elaborate hierarchial construction. The difficulties in raising the lower bound give hope that perhaps the answer to the original question is positive. Aside from the very strict lower bound on advice size for optimal algorithms, the question of lower bound tradeoff between the advice size and competitive ratio is interesting on itself and remains widely open.

References

1. Asahiro, Y., Miyano, E., Miyazaki, S., Yoshimuta, T.: Weighted nearest neighbor algorithms for the graph exploration problem on cycles. Information Processing Letters 110(3), 93–98 (2010)
2. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the Advice Complexity of the k-Server Problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the Advice Complexity of Online Problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis, vol. 2. Cambridge University Press (1998)
5. Dereniowski, D., Pelc, A.: Drawing maps with advice. J. Parallel Distrib. Comput. 72(2), 132–143 (2012)
6. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. ITA 43(3), 585–613 (2009)
7. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. 412(24), 2642–2656 (2011)
8. Flocchini, P., Mans, B., Santoro, N.: Sense of direction in distributed computing. Theor. Comput. Sci. 291(1), 29–53 (2003)
9. Fraigniaud, P., Gavoille, C., Ilcinkas, D., Pelc, A.: Distributed computing with advice: information sensitivity of graph coloring. Distributed Computing 21(6), 395–403 (2009)
10. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Impact of memory size on graph exploration capability. Discrete Applied Mathematics 156(12), 2310–2319 (2008)
11. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Communication algorithms with advice. J. Comput. Syst. Sci. 76(3-4), 222–232 (2010)
12. Fraigniaud, P., Korman, A., Lebhar, E.: Local mst computation with short advice. Theory Comput. Syst. 47(4), 920–933 (2010)
13. Fusco, E.G., Pelc, A.: Trade-offs between the size of advice and broadcasting time in trees. Algorithmica 60(4), 719–734 (2011)
14. Gutin, G., Punnen, A.P.: The Traveling Salesman Problem and Its Variations. Springer, Heidelberg (2002)
15. Hromkovič, J., Královič, R., Královič, R.: Information Complexity of Online Problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
16. Hurkens, C.A., Woeginger, G.J.: On the nearest neighbor rule for the traveling salesman problem. Operations Research Letters 32(1), 1–4 (2004)
17. Ilcinkas, D., Kowalski, D.R., Pelc, A.: Fast radio broadcasting with advice. Theor. Comput. Sci. 411(14-15), 1544–1557 (2010)
18. Kalyanasundaram, B., Pruhs, K.R.: Constructing competitive tours from local information. Theoretical Computer Science 130(1), 125–138 (1994)
19. Megow, N., Mehlhorn, K., Schweitzer, P.: Online Graph Exploration: New Results on Old and New Algorithms. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 478–489. Springer, Heidelberg (2011)
20. Miyazaki, S., Morimoto, N., Okabe, Y.: The online graph exploration problem on restricted graphs. IEICE Transactions on Information and Systems 92(9), 1620–1627 (2009)
21. Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M.: An analysis of several heuristics for the traveling salesman problem. SIAM J. Comput. 6(3), 563–581 (1977)

Asynchronous Exploration of an Unknown Anonymous Dangerous Graph with $O(1)$ Pebbles*

Balasingham Balamohan[1], Stefan Dobrev[2], Paola Flocchini[1], and Nicola Santoro[3]

[1] EECS, University of Ottawa, Ottawa, Canada
[2] Institute of Mathematics, Slovak Academy of Sciences, Bratislava, Slovak Republic
[3] School of Computer Science, Carleton University, Ottawa, Canada

Abstract. We consider the a team of asynchronous agents that must explore an unknown graph in presence of a black hole, a node which destroys all incoming agents without leaving any observable trace. Communication is achieved using pebbles that an agent can pick up, carry, and drop. It is known that, when the graph is unknown, $\Delta+1$ agents are necessary, and solutions exist with those many agents, using a total of $O(\log \Delta)$ pebbles, where Δ is the max node degree. On the other hand, it is also known that if the agents have a map of the graph, the problem can be solved with $O(1)$ pebbles in total, without increasing the size of the team. In this paper we address the question of whether it is possible to locate the black hole using $O(1)$ pebbles even if the graph is unknown, and, if so, with how many agents. We first prove that with $O(1)$ pebbles, $\Delta + 1$ agents are *not* sufficient. We next prove that, regardless of the team size, 2 pebbles are *not* sufficient. We then show that these bounds are *tight* presenting a protocol that allows to locate a black hole in an unknown anonymous graph with only 3 pebbles and $\Delta + 2$ agents.

1 Introduction

The Problem. *Black hole search* (BHS) refers to the graph exploration problem by a team of mobile agents when the graph is dangerous for the agents; the nature of the danger is the presence of harmful node, called *black hole*, which destroys all incoming agents without leaving any observable trace. A team of identical agents, executing the same protocol and starting from a single node called homebase, solves the black hole search problem if at least one agent survives, and all surviving agents within finite time discover the location of the black hole (e.g., know the edges leading to the black hole).

This problem has been extensively investigated, and the goal has been to understand which factors influence the solvability of the problem and its complexity, when solvable. The first consideration is the amount of synchronization

* Research partially supported by NSERC Discovery Grants, and by Dr. Flocchini URC.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 279–290, 2012.

provided by the system. In a *synchronous* system, clearly two agents suffice, and the focus has been on the time complexity of the solutions (e.g., see [6,12–14]).

In this paper, we are interested in solving the problem without any assumptions on time, that is in *asynchronous* systems. For such systems, the majority of investigations are based on *whiteboard* model of communication and synchronization: Each node provides shared memory accessible by fair mutual exclusion that can be used for communication among agents (e.g., see [1,4,8,9,11]). The whiteboard model is very powerful, providing not only direct and explicit communication but also act as mechanisms for leader election when agents are co-located and FIFO capabilities even when the network is not so. An alternative is the *token* model used in early investigations in graph exploration. It uses identical pebbles that an agent can hold, carry while moving, place on nodes, pick up from a node; no other marking on nodes for communication (e.g., see [7,10,15]).

The next distinction is whether the graph is known (i.e., the agents have a map) or unknown (and possibly *anonymous*); note that some information must however be available for the problem to be solvable; in particular, some metric information such as the number n of nodes and the number m of links, or the number of safe links (i.e., not leading to the black hole) is necessary for termination. Let us assume that such information is available to the agents. It is known that, both with tokens and with whiteboards, 2 agents suffice if they have a map of the graph; on the other hand, if the graph is a priori unknown to the agents, at least $\Delta(G)+1$ agents are necessary. Indeed size-optimal solutions exist both with map and in unknown anonymous graphs, both with tokens and with whiteboards [7,8,10].

The token model is generally viewed as less taxing of the system resources than whiteboards, and has been commonly employed in the exploration of safe graphs; it give raise to additional research questions, in particular: How many pebbles are needed ? Can pebbles be placed only on nodes (pure token model) or also on edges (hybrid token model) ? etc. These questions focus on how much *space* is required from the system by an synchronous token-based solution.

These questions are also relevant because of the relationship between solutions in the two models. In fact, any protocol which uses at most t pebbles in the pure token model, can be directly implemented using whiteboards of size at most $\lceil \log t \rceil$ bits; note that h pebbles in the hybrid model correspond to $t = h\Delta$ pebbles in the pure token model. This measure t, which we call *token load*, is clearly important in that it determines the usability of token-protocols in the whiteboard model, and provides a simple mechanism to transfer complexity results from one setting to the other. (Note that the transfer works also in the other direction: any solution which uses whiteboards of size s bits can be implemented with a token load at most $t = n2^s$.) Thus for example, the recent token-based solution for graphs of known topology with two agents and two pure tokens [10], implies a 2-agents solution using 1-bit whiteboards.

The fact that it is possible to locate a black hole using $O(1)$ tokens when the map of the graph is available [10] opens immediately the question of whether a similar result holds also if no map is available to the agents. In other words, is it

possible locate the black hole in a graph of unknown topology using a constant number of pebbles ? and if so, under what conditions ?

The current results for unknown graphs are not very useful and provides no hints. The existing whiteboard solution uses $O(\log n)$ bits whiteboards [8], implying a token solution that uses $t = O(n^2)$ pebbles; the existing token-based solution uses $t = O(\Delta^2)$ pebbles [7]. Similar questions have been recently raised, in the case of synchronous rings and synchronous tori [2,3].

The main contribution of this paper is to provide a definite answer to those questions. We first examine how many agents are needed to locate a black hole without a map of the graph. We prove the unexpected negative result that if $t = O(1)$, then $\Delta + 1$ agents are *not* sufficient. That is, unlike the case when the graph is known, the minimum team size required without a map can not be achieved if the number of tokens is bounded. We then prove that, regardless of the team size $t = 2$ pebbles are *not* sufficient. We then show that these bounds are tight: it is indeed possible for a team of $\Delta + 2$ agents to locate a black hole in an unknown graph using only 3 pebbles. The solution protocol employs as primitives a series of simple but novel token-based communication protocols; the messages communicated using tokens are all of polynomial length.

2 Definitions and Basic Constraints

The network environment is a simple undirected edge-labelled graph $G = (V, E)$ of $|V| = n$ nodes and $|E| = m$ edges. The network is *anonymous*; that is, the nodes have no distinct identifiers that can be used in the computation. At each node $x \in V$ there is a distinct label (called port number) associated to each of its incident links (or ports); without loss of generality, we assume that the labels at $x \in V$ are the consecutive integers $\#1$, $\#2$, $\ldots$, $\#d(x)$, where $d(x)$ denotes the degree of x; we denote by $\Delta(G)$ (or simply Δ) the maximum node degree in G. If $(x, y) \in E$ then x and y are said to be neighbours.

Operating in G is a team $\mathcal{A}$ of anonymous (i.e., identical) agents. The agents obey the same set of behavioral rules (called *algorithm* or protocol). Initially, they are all in the same state, and enter the network from the same node h, called *homebase*, but not necessarily at the same time. The agents can move from node to neighboring node in G. Links are not necessarily FIFO.

The agents operate within the so-called pure *token model*: the only mechanism available to the agents for coordination, control and communication is by means of identical *pebbles* that each agent is capable to pick up from a node, hold, carry while moving, and drop in the center of a node. We denote by t the *token load*, that is the total number of pebbles available in the system. Initially, the tokens can be either held by (some of) the agents or located on the homebase; without loss of generality, we assume the latter. Agents can see the tokens placed on the node they are currently visiting, but not the other agents currently there; i.e., agents are *invisible* to each other. When active at a node, an agent may attempt to access the pebbles at that node (to determine the number, to drop pebbles, or to pick up pebbles); attempts by different agents at the same node

are resolved in fair mutual exclusion. In other words, every agent attempting an access, will be granted one within finite time. The agents are *asynchronous* in the sense that every action they perform (moving, computing, pebble drop and pick) takes an unpredictable (but finite) amount of time; similarly, the interval between activities is finite but unpredictable. The agents do not have a map of G; they might however have some metric information, as discussed later.

The network contains a *black hole*, a node $Bh \in V$ that destroys any incoming agent without leaving any trace of that destruction. The goal of a *black hole search* algorithm $\mathcal{P}$ is to identify the location of Bh; that is, at least one agent survives, and all the surviving agents within finite time know the edges leading to the black hole. Knowledge of the location of the black hole is *exact* if all and only the links leading to the black hole have been determined; it is *partial* if all the links leading to the black hole have been determined (i.e., some links might be incorrectly suspected to lead to the black hole). Termination of a solution protocol is *explicit* if within finite time all surviving agents enter a terminal state and have the same information on the location of the black hole in the graph. We are obviously interested in protocols that explicitly terminate with exact information. Some basic limitations are well known [8,9]:

Lemma 1. *(1) If G has a cut vertex different from the homebase, the* BHS *problem is unsolvable. (2) In the absence of other topological knowledge, if n is not known, the* BHS *problem is unsolvable.*

As a consequence, we assume that G remains connected once BH is removed, and that n is known to the agents. Knowledge of n implies that an algorithm could terminate as soon as $n - 1$ safe nodes have been visited. However, due to asynchrony, explicit termination (even with just partial knowledge) may be impossible, as expressed by the following simple observations:

Lemma 2. *In absence of any other topological knowledge, explicit termination is not possible (1) if only n is known; (2) if only m is known; (3) if only n and an upper bound $\Delta' \geq \Delta$ are known.*

On the other hand:

Lemma 3. *If n and m are known, explicit termination with exact knowledge is possible.*

As a consequence, to ensure explicit termination, we assume that also m is known. Our algorithm can be easily modified for the case in which m is unknown; in that case, the termination will be implicit.

3 Impossibilities and Lower Bounds

Recall that $\Delta + 1$ agents are sufficient to locate the BH if the token load is a function of the size of the network, and thus a priori unbounded [7,8]. In this

section we show that if the total number of tokens is bounded, $\Delta + 1$ agents no longer suffice.

The proof of this result is based on a game between an algorithm and the adversary. The adversary has the power to choose the graph, the port labeling and the asynchronous timing of the edges; the algorithm specifies agent movement; w.l.o.g. we can assume the steps of the algorithm and adversary alternate. At any moment of time, the agents can be classified as *free* and *blocked*. The free agents are located at the nodes, or traversing an already explored edge (an edge is unexplored if no agent has so far crossed it in either direction). The algorithm specifies in its step the movement of the free agents located at the nodes; w.l.o.g. we may assume the algorithm specifies movement (the port taken) of at most one free agent in its step. The *blocked* agents are the agents which are in transit over the yet unexplored edges. In its step, the adversary lets the free agents crossing edges reach their destinations. Furthermore, if the algorithm specifies that an agent enters an unexplored edge with a given label, the adversary chooses which of the incident unexplored edges the agent enters and assigns that label to the edge. Finally, the adversary can unblock a blocked agent and allow it to reach its destination.

The adversary is limited by the requirement that it can indefinitely block only the agents crossing the links entering the BH. However, due to asynchrony, the unknown topology and the unknown location of the BH, this means that the only case when the adversary cannot block the blocked agents indefinitely is if there are at least $\Delta + 1$ blocked agents or there are two blocked agents traveling on edges leaving the same node. In such case, the algorithm can wait until the adversary unblocks one of the blocked agents; in all other cases, the algorithm must specify an agent movement in its step. We can now state:

Theorem 1. *When the overall number t of available tokens is less than a constant C, black hole search is unsolvable by $\Delta + 1$ agents without a map, even if the number of nodes and edges is known.*

Proof. By contradiction, assume $\Delta+1$ agents are sufficient. The unknown graph G in which the agents must locate the black hole is one of G_a, G_{b_1} or G_{b_2} shown in Figure 3, where HB is the homebase, and BH is the black hole; the choice of which graph G really is is up to the adversary. We now show that, even if the agents have the map of these three graphs, the uncertainty about which one G really is allows the the adversary to send all $\Delta + 1$ agents to the BH.

Starting from the homebase $HB = x_0$, whenever the algorithm sends an agent over the first unexplored link from node x_i $(0 \leq i \leq \Delta - 2)$, the adversary blocks it. Since G might be graph G_a, the algorithm must send an agent also over the second unexplored link incident on i. At this point, the adversary is forced to unblock one of these two links; it will unblock the link leading to node x_{i+1}. Hence, within finite time, $\Delta - 1$ agents are blocked and an agent reaches node $x_{\Delta-1}$.

The two free agents must now try to explore the remainder of the graph, since the adversary might have chosen $G = G_a$. In doing so, the adversary can

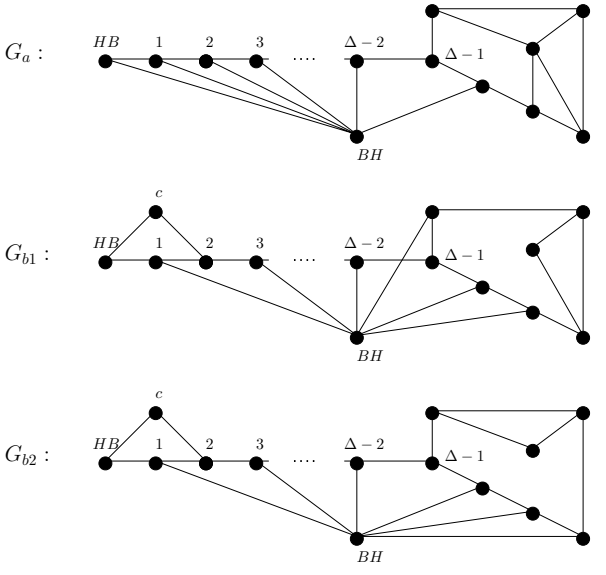


Fig. 1. The graphs used in Theorem 1

force both of them to explore a link leading to the black hole by choosing either $G = G_{b1}$ or $G = G_{b_2}$, depending on the way these two agents try to explore the remainder of the graph. Notice that at this point, all agents are blocked on links leading to the black hole, except for the two agents traversing the edges leading to node c. The adversary then unblock them; to finish the exploration of the graph. these two agents must reach node $x_{\Delta-1}$ and go beyond.

Regardless of the choice, $G = G_{b1}$ or $G = G_{b_2}$, made by the adversary, at each of the nodes $x_3, x_4, x_{\Delta-2}$ on the path to node $x_{\Delta-1}$, there is a link leading to the black hole, which the two agents must avoid. Since the algorithm can place only C tokens on the Δ nodes $c, x_0, ..., x_{\Delta-2}$, it can encode at most $O(\Delta^C)$ possibilities on how to safely move from node x_3 to node $x_{\Delta-1}$. However, the safe ports from all but one of the nodes x_3 to node $x_{\Delta-1}$ must have been encoded by the algorithm so not to fail; but such an information requires $2^{\Delta-5}$ different configuration of tokens. This means that, for large enough Δ, the adversary can force also these two agents to enter the black hole while they are moving from node x_3 to node $x_{\Delta-1}$.

As a consequence, at least $\Delta+2$ agents are required to locate the black hole. We now focus on the needed token load t, i.e., how many tokens are really necessary.

Theorem 2. *With two tokens initially stored at the HB and no additional tokens, no team of anonymous identical agents can locate the BH in an unknown network, even if the number of nodes and edges is known.*

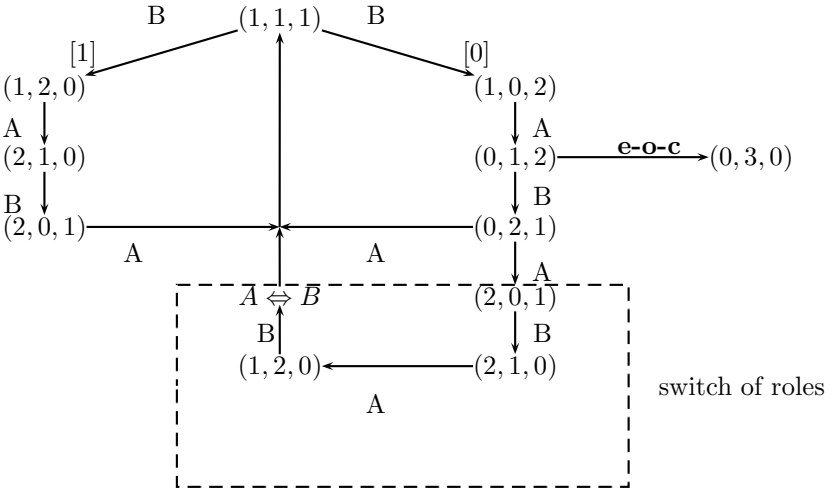


Fig. 2. Module Comm

4 Possibility and Upper Bounds

4.1 Communication and Coordination Using Tokens

A basic module of our solution protocol is a simple novel technique that uses three tokens to communicate finite but arbitrary information (a sequence of bits) between two agents: A and B. The algorithm uses two types of communication: with and without reply. All communication takes place in the home base. Let (x, y, z) denote the configuration of tokens in which A has x tokens, B has z tokens, and the homebase contains y tokens. Let $\mathcal{S} = b_1 b_2 \cdots b_r$ be a sequence of r bits to be transmitted from B to A. First of all, to unambiguously detect *termination*, the sequence will be encoded as: $\mathcal{S}' = 1b_1 1b_2 \cdots 1b_{r-1} 1b_r 0$ so that a zero in odd position will indicate termination. Communication always starts from configuration $(1, 1, 1)$.

The communication protocol (Module Comm) is as follows (see Figure 2): if the bit to be communicated is "0", B picks up a token creating configuration (1,0,2); if instead the bit is "1", B drops the token (conf. (1,2,0)). When agent A understands what has been communicated, A drops its token if the bit communicated is "0" (conf. (0,1,2)), it picks one up otherwise (conf. (2,1,0)) acknowledging reception of the information. At this point, three different actions are possible depending on whether *i*) the sequence is not terminated, *ii*) the sequence is terminated (the last bit is a zero in odd position) and no reply is required, *iii*) the sequence is terminated and a reply is required. In the first case B drops a token (conf. (0,2,1)) or picks one up (conf. (2,0,1)) depending on the last transmitted bit; then A creates (1,1,1) by picking up (or dropping) a token, allowing B to continue the transmission of the sequence. In case *ii*)

the communicated bit is surely zero; B drops both tokens reaching configuration (0,3,0) which indicates end of communication. Finally, in case *iii*) a protocol to switch role is executed which allows configuration (1,1,1) to be created by B, so that A can start its reply (following the same protocol with switched roles). More precisely, in a transmission with reply, B drops one token (conf. (0,2,1)); A starts the protocol for switching roles by grabbing both tokens. Once this is observed by B, it drops its own token (conf. (2,1,0)); once this is observed by A, it drops one of its tokens (conf. (1,2,0)); finally B picks one up creating configuration (1,1,1) where A will be the transmitted and B the receiver (denoted in the figure by $A \Leftrightarrow B$), allowing the beginning of the reply communication.

4.2 Black Hole Search

Informal Description. Algorithm Black hole search is composed of four modules: Leader Election, Find Waiting Room, Synchonization and Search, and it starts with three tokens initially placed on the homebase.

Module Elect Leader is executed first so that a leader is elected and all the other agents (the *followers*) are notified. Once elected, the acknowledged leader (from now on simply called leader) places all three tokens on the homebase. For reasons that will become apparent later, the next goal of the agents is to identify a second safe node, besides the homebase: the *waiting room*. Note that if the home base has only one neighbour, it must be safe since we assumed that the removal of the black hole does not disconnect the graph. Should the home base have more than one neighbour, this node will be one connected though either port 1 or port 2. (Module Find Waiting Room). As soon as the leader determines the location of the waiting room, it synchronizes the other agents (Module Synch) so they can start the black hole search (Module Search).

The actual search is coordinated by the leader, which gives instructions to the other agents indicating which new link to explore (if any) or to wait in the waiting room (if all known unexplored links are being explored). The agents explore new links and, if survive, report back their discoveries. Notice that since the graph is anonymous, the newly reported node might actually be already in the map without anybody knowing; this has to be verified.

Agents waiting in the waiting room are brought back to the homebase by the leader (by appropriate use of tokens in the waiting room) if there are new links in need of exploration.

Leader Election. All k agents are initially in state *Candidate*. A candidate seeing three tokens on the homebase picks one up and becomes *Leader* setting a counter to 1. A candidate seeing two tokens on the homebase picks one up and becomes a *Follower*. The Leader seeing a single token on the homebase picks up the token. A follower with one token, seeing none on the homebase drops the token and becomes an *Acknowledged Follower*. The leader with two tokens when it sees a single token drops a token and increment the counter by one to keep track of the number of acknowledged followers. When the counter is equal to k the leader becomes the *Acknowledged Leader*; at this point each agent knows

whether it is the leader or not. In the following, once this protocol has been executed, we will refer to the Acknowledged Leader simply as "leader", and to an Acknowledged Follower as "follower".

Determining the Waiting Room. The goal of Module FIND WAITING ROOM is to identify a safe node adjacent to the homebase, and in the process to "register" all followers.

Two coordination protocols are used in this phase: INSTRUCTF1ANDF2 and INSTRUCTOTHERS. Both of them are started with three tokens (conf. (0,3,0)) and with a follower picking up two tokens (conf. (0,1,2)). The first two followers executing this are special and the leader proceeds according to Protocol INSTRUCTF1ANDF2 : the first follower ($F1$) will be assigned the task to explore the first neighbour of the homebase, the other ($F2$) will have to explore the second. The protocol (see Figure 3 a)) involves a sequence of tokens manipulations between the leader and the follower until they are ready to talk (conf. (1,1,1)). At this point the leader starts Module COMM to make the follower aware of its role: to be $F1$ if it is the first, or $F2$ if it is the second, and to explore the first (resp. second) neighbour of the homebase. When communication is over the follower is registerd and three tokens are again present on the homebase. Note that three tokens are never on the home base during the execution of INSTRUCTF1ANDF2 except in the initial and in the final configuration.

Each subsequent unregistered follower, upon seeing three tokens (conf. (0,3,0)) on the home base, picks up two tokens (conf. (0,1,2)); the leader in this case, proceeds according to Protocol INSTRUCTOTHERS (see Figure 3 b)) which terminates with the follower being registered and with either three or two tokens in the home base (depending on whether the waiting room is known or not). This protocol does not involve actual communication, but only some token manipulation. At least one of agents F_1 and F_2 will return from their exploration. If successful, they use different methods to communicate to the leader, F_1 "communicates" to the leader that the first neighbour is safe by grabbing three tokens from the homebase (when it sees them available) and bringing them to the newly explored node. An empty home base can only mean that F_1 was safely back; hence, eventually the leader learns that F_1 has returned safely. If $F2$ returns safely, it cannot use the same method to communicate because it would lead to ambiguity. Instead, F_2 picks up 2 tokens (when it sees 3 available in the homebase) and starts protocol INSTRUCTOTHERS and registers again. The leader might not understand right away that $F2$ is back unless $k-1$ agents have already registered; it will do so however when eventually it will record k registrations. Hence, eventually the leader learns that F_2 has returned safely. If $F1$ returns before the leader has recorded k registrations, the leader selects the first neighbour of the home base to be the waiting room; otherwise, upon recording k registrations, the leader selects the second neighbour to be the waiting room.

Synchronization. Protocol SYNCH starts once the waiting room is determined and allows registered followers to synchronize with the leader for the SEARCH phase. The protocol is enabled by the leader creating configuration $(1, 2, 0)$. It

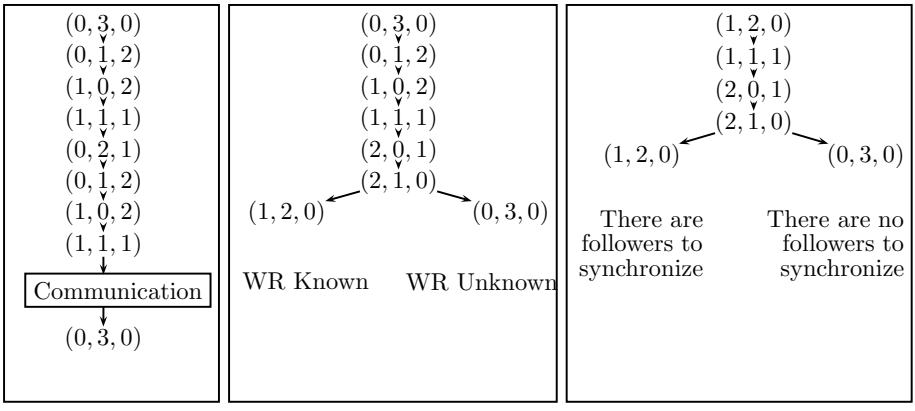


Fig. 3. Token configurations for the main modules. The first action is always taken by the follower.

is started by a registered follower not yet synchronized; depending on whether or not there are other registered followers to be synchronized, it terminates in configuration $(1, 2, 0)$ (enabling the protocol to be executed again) or $(0, 3, 0)$ (see Figure 3 *c*)) After executing protocol Synch, a follower becomes *synchronized*.

Search. The search is coordinated by the leader, which will create, maintain and update a partial map until the location of the black hole is determined. To do so, the leader gives instructions to the followers indicating to each of them which node to explore next. Such instructions are given in the form of paths to be taken to explore a new link, possibly leading to an unexplored node. A follower receiving such an instruction, follows the path and, if successful, comes back to the leader to report its findings: the safeness of the node and its degree (i.e. the number of nodes needing exploration). Upon receiving this information, the leader needs to update the map. Notice that since the graph is anonymous, the newly explored node might actually be already in the map without anybody knowing. To verify whether this is indeed a new node, the leader goes to the node, it places a token on it, and it traverses the current safe portion of the graph (i.e., the current map). If the token is found during the traversal it means it was already explored; otherwise the leader updates the map.

When giving instructions to a follower, if all links outgoing from the known safe area of the network are under exploration, the leader instructs the follower to move to the waiting room. Thus, it is possible that some agents are in the waiting room when an agent returns from exploring a link. If some work becomes available (i.e., new links to explore are reported), the leader goes to the waiting room to call back some agents. More precisely, the leader goes to the waiting room, places three tokens there, signalling to those waiting that there is work to be done. The first agent in the waiting room to notice, picking up the three

tokens, brings them to the homebase, and places two of them there. At this point the leader moves to the homebase as well, and picks up one of the two tokens reaching a situation where the two agents can communicate.

Any communication is achieved by Module Comm; communication is always with reply and is started by a follower picking up one token when seeing three. The leader seeing two tokens understands that the follower needs to talk, picks up one token and the conversation starts with one token on the homebase, one with the leader', and one with the follower. In the very first such communication, a follower receives also an ID from the leader.

Note that, because of asynchrony, the execution of Modules Search and Find Waiting Room may be interleaved; the communication protocols is carefully designed so to avoid interference between agents in different stages. For example, while the leader is coordinating the search, some agents (unregistered followers) might execute InteractOthers when they see three tokens available. The leader recognizes the situation by the fact that they pick up two tokens instead of one, and proceeds accordingly. Even $F1$ might come back safe while the leader is already executing the Search with all the other followers. In this case $F1$ grabs three tokens when it sees them available. The leader recognizes the situation as finding an empty home base has only this particular meaning, it then simply goes to the first neighbour of the home base to take the tokens back and puts them on the home base. This way, also $F1$ can start the Search phase.

4.3 Correctness and Complexity

We define as *active interaction* the action of an agent picking up or dropping one or more tokens. The following Lemmas hold:

Lemma 4. *Two agents can exchange arbitrary information with each other, starting and ending with three tokens on the homebase.*

Lemma 5. *Algorithm* Elect Leader *terminates in finite time and all agents eventually start Algorithm* Find Waiting Room*.*

Lemma 6. *In finite time, the* Leader *will know that one between the first or second neighbour of the homebase is safe.*

Let C^t be the set of registered followers that at time t are still not synchronized, D^t be the set of d unregistered followers, and W^t be the set of all followers. During search let A^t be the set of a^t links under exploration (through which no agent has returned yet), and B^t be the set of b^t unexplored links (i.e., through which no agent has exited). When no ambiguity arises, we shall omit the time in the notation.

Lemma 7. *Let a safe waiting room be identified by the leader. In finite time all followers start Algorithm* Search*.*

Based on the above Lemmas we can conclude:

Theorem 3. *Algorithm* Black Hole Search *solves* Bhs*.*

References

1. Balamohan, B., Flocchini, P., Miri, A., Santoro, N.: Improving the Optimal Bounds for Black Hole Search in Rings. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 198–209. Springer, Heidelberg (2011)
2. Chalopin, J., Das, S., Labourel, A., Markou, E.: Black Hole Search with Finite Automata Scattered in a Synchronous Torus. In: Peleg, D. (ed.) DISC 2011. LNCS, vol. 6950, pp. 432–446. Springer, Heidelberg (2011)
3. Chalopin, J., Das, S., Labourel, A., Markou, E.: Tight Bounds for Scattered Black Hole Search in a Ring. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 186–197. Springer, Heidelberg (2011)
4. 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)
5. Czyzowicz, J., Dobrev, S., Královič, R., Miklík, S., Pardubská, D.: Black Hole Search in Directed Graphs. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 182–194. Springer, Heidelberg (2010)
6. Czyzowicz, J., Kowalski, D.R., Markou, E., Pelc, A.: Complexity of searching for a black hole. Fundamenta Informaticae 71(2-3), 229–242 (2006)
7. Dobrev, S., Flocchini, P., Královic, R., Santoro, N.: Exploring an Unknown Graph to Locate a Black Hole Using Tokens. In: Navarro, G., Bertossi, L., Kohayakawa, Y. (eds.) TCS 2006. IFIP, vol. 209, pp. 131–150. Springer, Boston (2006)
8. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. Distributed Computing 19(1), 1–19 (2006)
9. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. Algorithmica 48, 67–90 (2007)
10. Flocchini, P., Ilcinkas, D., Santoro, N.: Ping pong in dangerous graphs: Optimal black hole search with pebbles. Algorithmica 62(3-4), 1006–1033 (2012)
11. Flocchini, P., Kellett, M., Mason, P., Santoro, N.: Searching for black holes in subways. Theory of Computing Systems 50(1), 158–184 (2012)
12. Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Hardness and approximation results for black hole search in arbitrary networks. Theoretical Computer Science 384(2-3), 201–221 (2007)
13. Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Approximation bounds for black hole search problems. Networks 52(4), 216–226 (2008)
14. Kosowski, A., Navarra, A., Pinotti, C.M.: Synchronous black hole search in directed graphs. Theoretical Computer Science 412(41), 5752–5759 (2011)
15. Shi, W.: Black hole search with tokens in interconnected networks. In: 11th Int. Symp. on Stabilization, Safety, and Security of Distr. Sys (SSS), pp. 670–682 (2009)

Time of Anonymous Rendezvous in Trees: Determinism vs. Randomization

Samir Elouasbi and Andrzej Pelc*

Département d'informatique, Université du Québec en Outaouais,
Gatineau, Québec J8X 3X7, Canada
{elos02,pelc}@uqo.ca

Abstract. Two identical (anonymous) mobile agents start from arbitrary nodes of an unknown tree and move along its edges with the goal of meeting at some node. Agents move in synchronous rounds: in each round an agent can either stay at the current node or move to one of its neighbors. We study optimal time of completing this rendezvous task. For deterministic rendezvous we seek algorithms that achieve rendezvous whenever possible, while for randomized rendezvous we seek *almost safe* algorithms, which achieve rendezvous with probability at least $1 - 1/n$ in n-node trees, for sufficiently large n.

We construct a deterministic algorithm that achieves rendezvous in time $O(n)$ in n-node trees, whenever rendezvous is feasible, and we show that this time cannot be improved in general, even when agents start at distance 1 in bounded degree trees. We also show an almost safe algorithm that achieves rendezvous in time $O(n)$ for arbitrary starting positions in any n-node tree. We then analyze when randomization can help to speed up rendezvous. For n-node trees of known constant maximum degree and for a known constant upper bound on the initial distance between the agents, we show an almost safe algorithm achieving rendezvous in time $O(\log n)$. By contrast, we show that for some trees, every almost safe algorithm must use time $\Omega(n)$, even for initial distance 1. This shows an exponential gap between randomized rendezvous time in trees of bounded degree and in arbitrary trees. Such a gap does not occur for deterministic rendezvous.

All our upper bounds hold when agents start with an arbitrary delay, controlled by the adversary, and all our lower bounds hold even when agents start simultaneously.

1 Introduction

Two identical mobile agents, initially located in two nodes of a network, move along links from node to node, and eventually have to meet in the same node at the same time. This task is known as rendezvous [1]. The network is modeled as an undirected connected graph, and agents traverse links in synchronous rounds.

* Supported in part by NSERC discovery grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 291–302, 2012.

Agents are mobile entities with unlimited memory; from the computational point of view they are modeled as copies of the same Turing machine. Agents do not know the topology of the network and they cannot mark the visited nodes in any way. We seek rendezvous algorithms that do not rely on the knowledge of node labels, and can work in anonymous graphs as well (cf. [1]). The importance of designing such algorithms is motivated by the fact that, even when nodes are equipped with distinct labels, agents may be unable to perceive them because of limited sensory capabilities, or nodes may refuse to reveal their labels, e.g., due to security or privacy reasons. On the other hand, we assume that edges incident to a node v have distinct labels in $\{0, \ldots, d-1\}$, where d is the degree of v. Thus every undirected edge $\{u, v\}$ has two labels, which are called its *port numbers* at u and at v. Port numbering is *local*, i.e., there is no relation between port numbers at u and at v. An agent entering a node learns the port of entry and the degree of the node. Note that, in the absence of port numbers, rendezvous is usually impossible, as all ports at a node look identical to an agent and the adversary may prevent the agent from taking some edge incident to the current node.

In this paper we focus attention on rendezvous in trees and our goal is to minimize rendezvous time. We consider both deterministic and randomized algorithms. For deterministic rendezvous we seek algorithms that achieve rendezvous whenever possible, while for randomized rendezvous we seek *almost safe* algorithms, which achieve rendezvous with probability at least $1 - 1/n$ in n-node trees, for sufficiently large n. It is well known (cf. e.g. [13]) that deterministic rendezvous with simultaneous start is possible if and only if the initial positions of the two agents are not symmetric, i.e., if there is no port-preserving automorphism of the tree that carries one node on the other.

Our Results. We construct a deterministic algorithm that achieves rendezvous in time $O(n)$ in n-node trees, whenever rendezvous is feasible, and we show that this time cannot be improved in general, even when agents start at distance 1 in bounded degree trees. More precisely, our algorithm guarantees rendezvous in linear time except for the special case of symmetric initial positions and simultaneous start, in which case deterministic rendezvous is impossible. We also show an almost safe algorithm that achieves rendezvous in time $O(n)$ for arbitrary starting positions in any n-node tree. We then analyze when randomization can help to speed up rendezvous. For n-node trees of known constant maximum degree and for a known constant upper bound on the initial distance between the agents, we show an almost safe algorithm achieving rendezvous in time $O(\log n)$. By contrast, we show that for some trees, every almost safe algorithm must use time $\Omega(n)$, even for initial distance 1. This shows an exponential gap between randomized rendezvous time in trees of bounded degree and in arbitrary trees. Such a gap does not occur for deterministic rendezvous.

All our upper bounds hold when agents start with an arbitrary delay, controlled by the adversary, and all our lower bounds hold even when agents start simultaneously.

Related Work. An extensive survey of randomized rendezvous in various scenarios can be found in [1]. Several authors considered the geometric scenario (rendezvous in an interval of the real line [5,14], or in the plane [2]).

For the deterministic setting a lot of effort has been dedicated to the study of the feasibility of rendezvous, and to the time required to achieve this task, when feasible. For instance, deterministic rendezvous with agents equipped with tokens used to mark nodes was considered, e.g., in [16]. Deterministic rendezvous of agents equipped with unique labels was discussed in [10,17]. (In the latter scenario, symmetry is broken by the use of the different labels of agents, and thus rendezvous is sometimes possible even for symmetric initial positions of the agents). Memory required by the agents to achieve deterministic rendezvous has been studied in [12,13] for trees and in [7] for general graphs. Memory needed for randomized rendezvous in the ring is discussed, e.g., in [15].

A natural extension of the rendezvous problem is that of gathering [9,11], when more than 2 agents have to meet in one location. In [9,11] gathering in the plane was considered and the authors assumed limited visibility, studying the deterministic scenario in [11] and the randomized scenario in [9]. In [19] the authors considered gathering of many agents with unique labels in networks.

Apart from the synchronous model used in this paper, several authors have investigated asynchronous rendezvous in the plane [6,11] and in network environments [4,8]. In the latter scenario the agent chooses the edge which it decides to traverse but the adversary controls the speed of the agent. Under this assumption rendezvous in a node cannot be guaranteed even in very simple graphs and hence the rendezvous requirement is relaxed to permit the agents to meet inside an edge. In [3] the authors study the memory size needed for time-optimal asynchronous rendezvous in trees, under a slightly different scenario: They do not allow rendezvous inside an edge, but for symmetric trees they allow that agents terminate not in one node but in two adjacent nodes.

2 Framework and Preliminaries

We consider trees with unlabeled nodes and labeled ports. An isomorphism between trees $T = (V, E)$ and $T' = (V', E')$, where V is the set of nodes of T and V' is the set of nodes of T', is a bijection $f : V \to V'$, such that for any $w, w' \in V$, w is adjacent to w' if and only if $f(w)$ is adjacent to $f(w')$. It preserves port numbering if for any $w, w' \in V$, the port number corresponding to edge $\{w, w'\}$ at node w is equal to the port number corresponding to edge $\{f(w), f(w')\}$ at node $f(w)$. An automorphism is an isomorphism of a tree on itself. A pair of distinct nodes u, v of a tree is called *symmetric*, if there exists an automorphism f preserving port numbering, and such that $f(u) = v$. Rooted trees are called *isomorphic*, if there exists an isomorphism from one to the other, preserving port numbering and carrying the root to the root.

We consider identical mobile agents traveling in trees with locally labeled ports. Unless stated otherwise, the tree and its size are a priori unknown to the agents. We assume that agents are copies A and A' of the same Turing machine

$\mathcal{A}$, starting at two distinct nodes v_A and $v_{A'}$, called the *initial positions*. We will refer to such identical machines as a *pair of agents*. Hence a pair of agents execute an identical algorithm. It is assumed that the internal clocks of a pair of agents tick at the same rate. The clock of each agent starts when the agent starts executing its actions. Agents start from their initial position with *delay* $\theta \geq 0$, controlled by an adversary. This means that the later agent appears at its starting position and starts executing its actions θ rounds after the first agent. Agents do not know which of them is first and what is the value of θ. The time of a rendezvous algorithm is the number of rounds since the start of the later agent until rendezvous.

Initial positions forming a symmetric pair of nodes are crucial for deterministic rendezvous. Indeed, if the initial positions are symmetric, then deterministic rendezvous is impossible for any pair of agents with simultaneous start, i.e, for $\theta = 0$, cf., e.g., [13]. We say that a pair of agents using a deterministic algorithm solve the rendezvous problem in a class of trees, if, for any tree in this class, the agents meet except for the above special case when rendezvous is impossible, i.e., if they achieve rendezvous from arbitrary initial positions when $\theta > 0$ and from arbitrary non-symmetric initial positions when $\theta = 0$. Note that agents do not know the value of θ and they do not know if their initial positions are symmetric, so the same rendezvous algorithm must cover both situations. For a pair of agents using a randomized algorithm, we say that they solve the rendezvous problem in a class of n-node trees, if the probability that both agents are eventually in the same node of the tree in the same round is at least $1 - 1/n$, regardless of the initial positions of the agents.

A *basic walk* in a tree T, starting from node v is a traversal of all edges of the tree ending at the starting node v and defined as follows. Node v is left by port 0; when the walk enters a node by port i, it leaves it by port $(i + 1)$ mod d, where d is the degree of the node. Every m-node rooted tree can be coded by a sequence of length $2(m-1)$ whose terms are consecutive port numbers encountered while performing the basic walk starting at the root (port 0 at every leaf is noted twice during its visit, at the entry and at the exit). Clearly, codes of rooted trees are identical if and only if these rooted trees are isomorphic. This yields a linear ordering of all non-isomorphic rooted trees, by the lexicographic ordering of their codes.

Consider any tree T and the following sequence of trees constructed recursively: $T_0 = T$, and T_{i+1} is the tree obtained from T_i by removing all its leaves. $T' = T_j$ for the smallest j for which T_j has at most two nodes. If T' has one node, then this node is called the *central node* of T. If T' has two nodes, then the edge joining them is called the *central edge* of T. A tree T with port labels is called *symmetric*, if there exists a non-trivial automorphism f of the tree (i.e., an automorphism f such that $f(u) \neq u$, for some $u \in V$) preserving port numbering. In a non-symmetric tree, every pair of nodes is non-symmetric. If a tree with port numbers has a central node, then it cannot be symmetric. If it has a central edge e, then it is symmetric if and only if the port numbers corresponding to e at both its extremities are the same and the subtrees resulting from the removal of e, rooted at these extremities, are isomorphic. These subtrees are called *halves* of

the tree T. (A code of each of them is obtained from a basic walk in it, starting at its root and skipping the port corresponding to e.)

3 Time of Deterministic Rendezvous

We first show that there exists a pair of agents that can deterministically solve the rendezvous problem in every tree, in time linear in the size of the tree. We assume that either $\theta = 0$ and the initial positions are non-symmetric, or that $\theta > 0$. (Otherwise, i.e., when initial positions are symmetric and start is simultaneous, rendezvous is impossible, as mentioned before.) The idea of the algorithm is the following. Each agent performs the basic walk, starting from its initial position. Upon its completion the agent has an isomorphic map of the tree with its starting position marked. If the tree is not symmetric, then the agent goes to an unambiguously chosen node and stops. Otherwise (in this case there is the central edge), each agent computes an integer label, based on its initial position, so that labels corresponding to non-symmetric positions are different. Then each agent goes to the closest endpoint of the central edge and starts executing phases of the form: ix traversals of the central edge and ix idle rounds, where x is its label, for $i = 1, 2, \ldots$, until rendezvous occurs. Below is the pseudo-code of the algorithm.

Algorithm Deterministic-RV
Perform the basic walk starting from initial position.
If the tree has a central node **then** go to the central node.
If the tree has a central edge and port numbers
    at both its endpoints are different **then**
    go to the endpoint corresponding to the larger port number.
If the tree has a central edge with equal port numbers,
    but the tree is not symmetric **then**
    go to the endpoint which is the root of the subtree
    with lexicographically larger code.
If the tree is symmetric **then**
    $x :=$ the number of steps of the basic walk starting from initial position until the first traversal of the central edge;
    go to the closest endpoint of the central edge via the shortest path;
        for $i = 1, 2, \ldots$ **do**
            traverse the central edge ix times;
            stay idle ix rounds.
        until rendezvous

Theorem 1. *Algorithm Deterministic-RV solves the rendezvous problem in an arbitrary tree, and takes time $O(n)$ in trees of size n.*

Proof. We first show that the algorithm is correct. After performing the basic walk, each agent knows which of the four "if" cases holds. If the central node exists, then it is unique and both agents meet there. If the tree has a central edge

and port numbers at both its endpoints are different, then the meeting endpoint is unambiguously defined and agents meet there. If the tree has a central edge with equal port numbers but the tree is not symmetric, then the two subtrees rooted at the endpoints of the central edge are not isomorphic and hence their codes are different. It follows that the root of the subtree with lexicographically larger code is unambiguously defined and both agents meet there.

Hence in the sequel we may assume that the tree is symmetric, i.e., both port numbers at the central edge are equal and the two subtrees rooted at the endpoints of the central edge are isomorphic. Consider two cases.

First assume that the initial positions are not symmetric. It follows that the numbers of steps of the basic walk starting from each of these positions until the first traversal of the central edge must be different. Hence the integers x computed by each agent are different. Call these integers x_1 and x_2 and assume, without loss of generality, that $x_1 > x_2$. Give the name a_j to the agent whose integer is x_j, for $j = 1, 2$. Let t be the round when the agent that started later the execution of the "do until rendezvous" loop started this loop. We show that by round $t + 4x_1$ rendezvous must occur. Indeed, this happens as soon as one of the agents is idle and the other agent moves. The worst case is when $x_1 = kx$ for some integer $k > 1$ and the agent a_1 starts the execution of the first turn of the loop in the same round in which agent a_2 starts the execution of the kth turn of the loop. Then they are both active for x_1 rounds and then both idle for x_1 rounds. Afterwards, agent a_1 is active for $2x_1$ rounds, while agent a_2 is active only for $(k + 1)x_2 < 2x_1$ rounds and then becomes idle. It follows that rendezvous occurs by round $t + 2x_1 + (k + 1)x_2 + 1 \leq t + 4x_1$.

Next assume that the initial positions of the agents are symmetric but $\theta > 0$. In this case the integer x computed by both agents is the same. Again, let t be the round when the agent that started later the execution of the "do until rendezvous" loop started this loop. Now we show that by round $t + 2x + 1$ rendezvous must occur. Now the worst case is when $t + x$ is the round when the earlier agent ends the activity part of its kth turn of the loop. Both agents are active for x rounds, then the earlier agent is idle for $(k + 1)x$ rounds, while the later agent is idle only for x rounds and then resumes activity. It follows that rendezvous must occur by round $t + 2x + 1$.

It remains to estimate the execution time of Algorithm Deterministic-RV. According to the definition we start counting time at the appearance of the later agent. The basic walk takes time $2(n-1)$. The integer x computed by each agent (in the case when the tree is symmetric) is at most n, as it can exceed by at most 1 the length of the basic walk in the half of the tree where the agent starts. The time to get to the central node or to the endpoint of the central edge is at most $n/2$. The time of the execution of the "repeat until rendezvous" loop is at most $4x_1 \leq 4n$. Hence the total time between the appearance of the later agent and rendezvous is at most $15n/2$. □

Remark. Algorithm Deterministic-RV is designed to solve the rendezvous problem in a deterministic setting, i.e., under the assumption that either the initial positions of the agents are not symmetric, or that their start is not simultaneous.

However, for the special case of simultaneous start, it can be modified to work for arbitrary starting positions and to perform a more demanding task: if the initial positions are not symmetric, then rendezvous is achieved, and if the initial positions are symmetric, then both agents stop and report that rendezvous is impossible. The modification is to stop the algorithm after $15n/2$ rounds if rendezvous is not achieved and report that rendezvous is impossible; n is the number of nodes of the tree which each agent learns after performing the basic walk. It follows from the proof of Theorem 1 that, if the initial positions are not symmetric, then rendezvous is achieved at the latest after $15n/2$ rounds of its execution by the later agent (or by both agents for simultaneous start). Hence the modification is correct for simultaneous start and the algorithm still works in time $O(n)$. For arbitrary delay θ such a strengthening is clearly impossible, as the later agent can appear arbitrarily late and the earlier agent can never know if rendezvous is unfeasible (because positions are symmetric and start is simultaneous) or if it is feasible but delayed because θ is large.

Can the rendezvous time $O(n)$ be improved by a deterministic algorithm? The answer is clearly "no" if the initial positions of agents are at distance $\Omega(n)$. However, the following result shows that the answer is also "no", even if the initial distance between the agents is 1, the agents start simultaneously, and even in the class of trees of bounded degree.

Proposition 1. *Every deterministic algorithm solving the rendezvous problem takes time $\Omega(n)$ for some instances where initial positions of agents are at distance 1 and agents start simultaneously. This holds even for the class of trees of bounded degree.*

Proof. Consider a n-node line, where n is even, oriented by having ports 0 and 1 at the endpoints of each edge. Place the initial positions of the agents in the two middle nodes. These positions are at distance 1 and are not symmetric. Consider any deterministic rendezvous algorithm A and assume that the agents execute it starting simultaneously. During the first $n/2-1$ rounds the history of both agents is identical, as none of them reaches any endpoint of the line. Hence, during these rounds they perform identical moves, which implies that the distance between them in each of these rounds remains 1. Hence the algorithm must use at least $n/2$ rounds. □

4 Time of Randomized Rendezvous

As previously noted, deterministic rendezvous is impossible, if the initial positions are symmetric and the start is simultaneous. Can this restriction be overcome by the use of randomization, and if so, is it possible to do it within time linear in the size of the tree? It turns out that the answer to both questions is "yes". This is done by the following randomized algorithm. If the tree has a central node, the agent goes there and stops. If the tree has a central edge, the agent goes to the closest endpoint of the central edge and in the next $\lceil \log n \rceil$ rounds flips a coin and stays idle or traverses the edge depending on whether the outcome is heads or tails.

Algorithm Randomized-RV
Perform the basic walk starting from initial position.
Compute the number n of nodes in the tree.
If the tree has a central node **then** go to the central node
else
    go to the closest endpoint of the central edge via the shortest path;
    repeat $\lceil \log n \rceil$ times or until rendezvous, whichever comes first:
        with probability 1/2 stay idle
        and with probability 1/2 traverse the central edge.

Theorem 2. *Algorithm Randomized-RV is almost safe, solves the rendezvous problem in an arbitrary tree, from arbitrary starting positions, with arbitrary delay, and takes time $O(n)$ in trees of size n.*

Proof. We first show that the algorithm is almost safe. If the tree has a central node, then rendezvous will occur at this node. Hence we may assume that the tree has a central edge. Consider the round t when the later agent, call it a, comes to the endpoint of the central edge; it starts flipping the coin in round $t+1$. In round t the other agent, call it b is already in one of the endpoints of the central edge. (If both agents start flipping the coin at the same time, we call them a and b arbitrarily.) We may assume that it is the other endpoint, otherwise the rendezvous is accomplished. In each of the rounds $t+1, \ldots, t+\lceil \log n \rceil$, or until rendezvous, agent a flips the coin and agent b either flips the coin as well or finished flipping and stays idle. Consider the event E that rendezvous did not occur during these rounds, i.e., in each of them the agents were at different endpoints of the central edge. Consider such a round r. Since in the beginning and at the end of round r the agents were at different endpoints, two cases are possible. The first case is that in round r both agents flipped the coin and they both moved or both stayed idle. The probability of this event is 1/2. The second case is that agent b already stopped flipping the coin and stayed idle, while agent a flipped the coin and stayed idle at the other endpoint. The probability of this event is also 1/2. It follows that the probability of the event E is $(1/2)^{\lceil \log n \rceil} \leq 1/n$, and hence the algorithm is almost safe.

We conclude by estimating the execution time of Algorithm Randomized-RV. Again we start counting time at the appearance of the later agent. The basic walk takes time $2(n-1)$. The time to get to the central node or to the endpoint of the central edge is at most n. The randomized part of the algorithm takes $\lceil \log n \rceil$ rounds. Hence the total time is at most $3n + \lceil \log n \rceil$. □

As in the deterministic case, randomized rendezvous cannot be achieved in sublinear time if the initial positions of the agents are at distance $\Omega(n)$. However, it turns out that for agents starting near each other, randomization can dramatically decrease rendezvous time. We have seen in Section 3 that deterministic rendezvous may require linear time even when agents start at distance 1 and even in the line. This should be contrasted with the situation when randomization is possible. Indeed, we will show that if agents start at a constant distance

D in a n-node tree of constant maximum degree Δ, and if parameters D, Δ, n are known to the agents, then almost safe randomized rendezvous can be accomplished in time logarithmic in the size of the tree. This is done by Algorithm Fast-Randomized-RV. Its idea is the following. Knowing parameters D and Δ the agent can find the upper bound $\alpha = \Delta^{D+1}$ on the size of any rooted subtree of depth D of the tree in which it operates. Then each agent divides all rounds since its start into consecutive segments of equal length 4α. In the first round of each segment it flips a coin. If the result is heads, the segment becomes active, otherwise it becomes passive. In an active segment the agent performs the basic walk in the subtree of depth D rooted at its initial position and then remains idle till the end of the segment. In a passive segment the agent remains idle for the entire duration of the segment. After the completion of $3\lceil \log n \rceil$ segments the agent stops. Below is the pseudo-code of the algorithm.

Algorithm Fast-Randomized-RV
$\alpha := \Delta^{D+1}$.
repeat $3\lceil \log n \rceil$ times or until rendezvous, whichever comes first:
    with probability 1/2 stay idle for 4α rounds
    and with probability 1/2 do
        perform the basic walk of the subtree of depth D
        rooted at initial position during z rounds and stay idle
        during the next $4\alpha - z$ rounds.

Theorem 3. *Algorithm Fast-Randomized-RV is almost safe and solves the rendezvous problem in an arbitrary n-node tree of constant maximum degree Δ, from arbitrary starting positions at constant distance D, with arbitrary delay, provided that agents know parameters D, Δ and n. It takes time $O(\log n)$ in trees of size n.*

Proof. Let $t_1 \leq t_2$ be the starting rounds of the two agents. Let $S_1, S_2, \ldots$ be the consecutive segments of rounds of length 4α for the first agent and let $T_1, T_2, \ldots$ be such consecutive segments of rounds for the second agent. Suppose that t_2 is in segment S_i. Since all segments are of the same length 4α, at least one of the following properties must hold. Either, for every natural number q, segment S_{i+q} and segment T_{1+q} have an overlap of length at least 2α, or for every natural number q, segment S_{i+1+q} and segment T_{1+q} have an overlap of length at least 2α. We show the analysis in the first case; the second case is analogous. The pairs of segments S_{i+q} and T_{1+q}, for $q = 0, 1, \ldots$ will be called *matched*.

Suppose that the segment S_{i+q} is passive and the segment T_{1+q} is active. This means that during the first $z \leq 2\alpha$ rounds of segment T_{1+q} the second agent performs the basic walk of the subtree of depth D rooted at its initial position, while the first agent stays idle at its initial position. Since the initial positions of the two agents are at distance D, the second agent will visit the initial position of the first agent during its basic walk and rendezvous will occur.

Consider a pair S_{i+q} and T_{1+q} of matched segments, for $q = 0, 1, \ldots, 3\lceil \log n \rceil - 1$. There are two cases. In the first case the activity or passivity of both segments is decided by coin flips and the probability of the event that the segment S_{i+q}

is passive and the segment T_{1+q} is active is 1/4. In the second case the first agent finished the execution of the algorithm (i.e., the segment S_{i+q} is passive) and the segment T_{1+q} is active with probability 1/2. Hence in both cases the probability of rendezvous during the segment T_{1+q} is at least 1/4. It follows that the probability that rendezvous does not occur after the $3\lceil \log n \rceil$ executions of the "repeat" loop is at most $(3/4)^{3\lceil \log n \rceil} = (27/64)^{\lceil \log n \rceil} < (1/2)^{\log n} = 1/n$, and hence the algorithm is almost safe. The estimate of the running time follows directly from the formulation of the algorithm, since the duration 4α of each segment is constant. □

Remark. It should be noted that the assumptions of the theorem can be slightly weakened. It is enough that agents know any constant upper bounds on the initial distance and on the maximum degree and any polynomial upper bound on the size of the tree.

To prove our final result we will use the following well known tool for proving lower bounds on the performance of randomized algorithms.

Lemma 1 (Yao's minimax principle [18]). *Let $0 < \epsilon < 1/2$. For any probability distribution $\mathcal{P}$ over the set of inputs, let $\mathcal{A}(\mathcal{P})$ denote the set of all deterministic algorithms that err with probability at most 2ϵ over $\mathcal{P}$. For $A \in \mathcal{A}(\mathcal{P})$, let $C(A, \mathcal{P})$ denote the expected running time of A over $\mathcal{P}$. Let $\mathcal{R}$ be the set of randomized algorithms that err with probability at most ϵ for any input, and let $T(R, I)$ denote the expected running time of $R \in \mathcal{R}$ on input I. Then, for all $\mathcal{P}$ and all $R \in \mathcal{R}$,*

$$\min_{A \in \mathcal{A}(\mathcal{P})} C(A, \mathcal{P}) \leq 2 \max_{I} T(R, I).$$

The standard application of the above lemma to lower bound proofs is the following. We construct a probability distribution over the set of inputs to a given problem, for which any deterministic algorithm that errs with probability at most 2ϵ has a large expected running time over this probability distribution. In view of the lemma, this implies that every randomized (Monte Carlo) algorithm that errs with probability at most ϵ for any input, must have large expected running time on some input.

First we consider the following auxiliary task which will be called Find-Treasure. There are k boxes numbered $0, 1, \ldots k-1$, one of which contains a treasure. Boxes can be opened one by one, in an arbitrary order, until the treasure is found. The running time of an algorithm solving the problem Find-Treasure is the number of boxes opened. Using Lemma 1, the following fact can be proved.

Lemma 2. *Any randomized algorithm for the Find-Treasure problem with k boxes, that errs with probability at most $1/\sqrt{k}$ has expected running time at least $k/16$ on some input, for sufficiently large k.*

Proof. An input of the Find-Treasure problem with k boxes is the number of the box containing the treasure. We use Lemma 1 with $\epsilon = 1/\sqrt{k}$ and with the uniform probability distribution $\mathcal{P}$. Consider any deterministic algorithm for the problem, that errs with probability at most $2/\sqrt{k}$, for the uniform distribution

over the inputs. For sufficiently large k, such an algorithm has to open at least $\lceil k/2 \rceil$ boxes. Let $a_1, a_2, \ldots, a_{\lceil k/2 \rceil}$ be the permutation of box numbers in the order the algorithm opens them. Hence the expected running time of the algorithm for the uniform distribution of inputs is at least $(1/k)(1+2+\cdots+\lceil k/2 \rceil) \geq k/8$. By Lemma 1, for any randomized algorithm that errs with probability at most $1/\sqrt{k}$ there exists an input for which this algorithm has expected running time at least $k/16$. □

We are now ready to prove our main negative result. Together with Theorem 3 it shows a significant difference between the impact of bounded maximum degree of the tree on the time of deterministic vs. randomized rendezvous. For deterministic rendezvous we have seen that the linear lower bound holds even on the line and even when both agents know that they are initially at distance 1. Hence the bounded maximum degree of the graph is not of any help in general. By contrast, for randomized rendezvous this restriction is crucial. While Theorem 3 showed that, for bounded degree graphs, randomized agents at initial distance 1 can solve rendezvous in logarithmic time, if they know parameters D, Δ and n, the following result shows that they need exponentially more time for some trees of unbounded degrees.

Theorem 4. *There exists a class of n-node trees of maximum degree $\lceil n/2 \rceil$, for all positive integers $n \geq 2$, such that every almost safe randomized algorithm that solves the rendezvous problem for these trees has expected running time $\Omega(n)$ on some n-node tree of this class, for any n, even if the initial positions of the agents are at distance 1, even when the agents know this and know the size of the tree in which they operate, and even when they start simultaneously.*

Proof. Let $n = 2k$; the construction for odd n is similar. The tree $T(n, i)$, for $i = 0, 1, \ldots, k-1$, is defined as follows. It has two adjacent nodes v and w with port number i at v and at w corresponding to the edge $\{v, w\}$. Moreover, both v and w are adjacent to $k-1$ leaves. Hence the tree $T(n, i)$ is composed of two stars whose centers are joined by an edge. The initial positions of the agents are at nodes v and w.

Consider a randomized rendezvous algorithm A on the tree $T(n, i)$ that errs with probability at most $1/n$. If rendezvous is accomplished, then at least one of the agents must have traversed the edge $\{v, w\}$. Hence the probability that a given agent traverses this edge is at least $1 - 1/\sqrt{n}$. This is equivalent to solving the Find-treasure problem with k boxes with error probability at most $1/\sqrt{n} \leq 1/\sqrt{k}$. By Lemma 2 the expected running time of such an algorithm is at least $k/16$ on some input, for sufficiently large k. It follows that, for sufficiently large n, the expected running time of algorithm A on the tree $T(n, i)$ is at least $k/16 \in \Omega(n)$, for some i. □

References

1. Alpern, S., Gal, S.: The theory of search games and rendezvous. Int. Series in Operations research and Management Science. Kluwer Academic Publisher (2002)
2. Anderson, E., Fekete, S.: Two-dimensional rendezvous search. Operations Research 49, 107–118 (2001)

3. Baba, D., Izumi, T., Ooshita, F., Kakugawa, H., Masuzawa, T.: Space-Optimal Rendezvous of Mobile Agents in Asynchronous Trees. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO 2010. LNCS, vol. 6058, pp. 86–100. Springer, Heidelberg (2010)
4. Bampas, E., Czyzowicz, J., Gąsieniec, L., Ilcinkas, D., Labourel, A.: Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 297–311. Springer, Heidelberg (2010)
5. Baston, V., Gal, S.: Rendezvous search when marks are left at the starting points. Naval Reaserch Logistics 48, 722–731 (2001)
6. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the Robots Gathering Problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1181–1196. Springer, Heidelberg (2003)
7. Czyzowicz, J., Kosowski, A., Pelc, A.: How to meet when you forget: Log-space rendezvous in arbitrary graphs. In: Proc. 29th Annual ACM Symposium on Principles of Distributed Computing (PODC 2010), pp. 450–459 (2010)
8. Czyzowicz, J., Labourel, A., Pelc, A.: How to meet asynchronously (almost) everywhere. In: Proc. 21st Ann. ACM Symp. on Discr. Algorithms (SODA 2010), pp. 22–30 (2010)
9. Degener, B., Kempkes, B., Meyer auf der Heide, F.: A local O(n2) gathering algorithm. In: Proc. 22nd Ann. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2010), pp. 217–223 (2010)
10. Dessmark, A., Fraigniaud, P., Kowalski, D., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica 46, 69–96 (2006)
11. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of Asynchronous Oblivious Robots with Limited Visibility. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 247–258. Springer, Heidelberg (2001)
12. Fraigniaud, P., Pelc, A.: Deterministic Rendezvous in Trees with Little Memory. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 242–256. Springer, Heidelberg (2008)
13. Fraigniaud, P., Pelc, A.: Delays induce an exponential memory gap for rendezvous in trees. ArXiv: 1102.0467v1 (2011)
14. Gal, S.: Rendezvous search on the line. Operations Research 47, 974–976 (1999)
15. Kranakis, E., Krizanc, D., Morin, P.: Randomized Rendez-Vous with Limited Memory. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 605–616. Springer, Heidelberg (2008)
16. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous in a ring. In: Proc. 23rd Int. Conf. on Distr. Computing Systems (ICDCS 2003), pp. 592–599 (2003)
17. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 599–608 (2007)
18. Yao, A.C.-C.: Probabilistic computations: Towards a unified measure of complexity. In: Proc. 18th Annual IEEE Conference on Foundations of Computer Science (FOCS 1977), pp. 222–227 (1977)
19. Yu, X., Yung, M.: Agent Rendezvous: A Dynamic Symmetry-Breaking Problem. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 610–621. Springer, Heidelberg (1996)

Randomized Rendezvous of Mobile Agents in Anonymous Unidirectional Ring Networks*

Shinji Kawai, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa

Graduate School of Information Science and Technology, Osaka University
{s-kawai,f-oosita,kakugawa,masuzawa}@ist.osaka-u.ac.jp

Abstract. We consider the rendezvous problem of multiple (mobile) agents in anonymous unidirectional ring networks under the constraint that each agent knows neither the number of nodes nor the number of agents. First, we prove for any (small) constant $p(0 < p \leq 1)$ that there exists no randomized algorithm that solves, with probability p, the rendezvous problem with (terminal) detection. For this reason, we consider the relaxed rendezvous problem, called the rendezvous problem without detection that does not require termination detection. We prove that there exists no randomized algorithm that solves, with probability 1, the rendezvous problem without detection. For the remaining cases, we show the possibility, that is, we propose a randomized algorithm that solves, with any given constant probability $p(0 < p < 1)$, the rendezvous problem without detection.

1 Introduction

Background and motivation. A *distributed system* consists of some computers (*nodes*) and communication links. In recent years, distributed systems have become large and design of systems has become more and more complicated. Because of this factor, *mobile agents* have received a lot of attention [1,2,3,4,5,6,7,8,9,10,11,12]. A (mobile) agent is an autonomous software that can move in the network with keeping some information. So, the agent is convenient for a task spreading over computers and, thus, it is expected to simplify the complicated design of the systems.

A distributed system with mobile agents is called a *mobile agent system*. In mobile agent systems, the *rendezvous problem* is known as one of the fundamental problems [1,2,3,4,5,6,8,9,10,11]. In the rendezvous (gathering) problems, multiple mobile agents dispersed in the network are required to meet at a single node. The rendezvous can be used to share information among all the agents or to synchronize behaviours of all agents.

In the rendezvous problem, initially agents have no knowledge of the network topology or other agents. This requires agents to move around the network and to

* This work is supported in part by Grant-in-Aid for Scientific Research ((B)2030012, (B)22300009, (B)23700056, (C)24500039) of JSPS.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 303–314, 2012.

determine the meeting node based on the collected information. The rendezvous problem can be easily solved if each node in a network has a unique identifier or ID: Each agent explores the network and terminates at the node with the smallest ID. However, such a unique ID may not be available for the agents in some cases. It may be prohibited to publish the node ID to agents for security reasons. Hence, it is important to design algorithms which work in anonymous networks.

Related works. In anonymous unidirectional ring networks, agents using the same deterministic algorithm cannot solve the rendezvous problem because of impossibility in breaking symmetry. The problem can be feasible with some additional assumptions such that agents can leave marks on nodes [4,5,9], can use randomness [1,8], knows an upper bound on the size of the network [6], the edge labeling is restricted [3,9], or the network topology is restricted [2]. Dieudonné and Pelc [6] consider *deterministic* algorithms for the rendezvous in arbitrary networks. They propose an algorithm for the rendezvous problem *with (terminal) detection* if agents know an upper bound on the size of the network. For the case that no upper bound is known to agents, they also propose an algorithm for the rendezvous problem *without detection*. They relax the problem to the rendezvous problem without detection, where all agents eventually gather and stop at the same node but without detecting the completion of the rendezvous, and propose an algorithm for the relaxed problem. In contrast, in this paper, we focus on algorithms using *randomness*. Randomized algorithms for the rendezvous problem have been studied [1,8,11]. Kranakis and Krizanc [8] propose a randomized algorithm based on random walk for the rendezvous of *two* agents in unidirectional ring networks. This algorithm requires no knowledge of the number of nodes and achieves rendezvous in $O(n^2)$ expected steps. Alpern et al. [1] propose a randomized algorithm for *two* agents, which is based on Coin Half Tour. This algorithm requires knowledge of the number of nodes and achieves rendezvous in $O(n)$ expected steps. However, for the problem with more than 2 agents in anonymous unidirectional rings, no randomized algorithm is known.

Our contribution. We consider the rendezvous problem of k ($k \geq 2$) agents in unidirectional ring networks. We also assume that each agent knows neither the number of nodes nor the number of agents. First, we prove for any (small) constant $p(0 < p \leq 1)$ that there exists no randomized algorithm that solves, with probability p, the rendezvous problem *with (terminal) detection*. This means we cannot design a randomized algorithm even if we require only small probability to achieve rendezvous. For this reason, we consider the rendezvous problem *without detection*. We prove that there exists no randomized algorithm that solves, with probability 1, the rendezvous problem without detection. For the remaining cases, we show the possibility, that is, we propose a randomized algorithm that solves, with any given constant probability $p(0 < p < 1)$, the rendezvous problem without detection.

2 Preliminaries

Network Model. A unidirectional ring network R is defined as 2-tuple $R = (V, E)$, where V is a set of nodes and E is a set of unidirectional links. We denote by $n(= |V|)$ the number of nodes. Then, ring R is defined as $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and $E = \{v_i, v_{(i+1) \bmod n} | 0 \leq i \leq n-1\}$.

Every node v_i has a *whiteboard* and any agent visiting node v_i can read from or write to the whiteboard. Ring R is *anonymous*, that is, each node has no ID. This means that an agent cannot distinguish two nodes when they have the same whiteboard contents.

Agent Model. We denote by k the number of agents. Let $A = \{a_0, a_1, \ldots, a_{k-1}\}$ be a set of agents. We model the agents as identical probabilistic finite automata $(S, \delta, s_{initial})$. The first element S is the set of states of the agent, which includes an initial state $s_{initial}$. Because every agent is identical, every agent initially starts with the same state $s_{initial}$. The second element δ is the state transition function δ:$S \times W \times RN \rightarrow S \times W \times M$, where W represents a set of states (or contents) of whiteboard, RN represents a set of random values, and $M = \{\texttt{move}, \texttt{stay}\}$ represents whether the agent makes movement or not in the step. The value `move` represents movement to the next node and `stay` represents stay at the current node. We assume that agents move instantaneously, that is, agents always exist at nodes (do not exist at links). This assumption is introduced for simplicity and does not lose generality even in the asynchronous model we consider because agents are asynchronously activated at nodes and are unaware of other agents at the same node. All agents have the same state transition function δ since they are identical.

System Configuration. In an agent system, (global) *configuration* c is defined as $(\mathcal{S}, \mathcal{W}, \mathcal{L})$, where $\mathcal{S} \in S^k$ represents states of agents, $\mathcal{W} \in W^n$ represents states of nodes (whiteboards), and $\mathcal{L} \in \{0, 1, \ldots, n-1\}^k$ represents *locations* of agents. The locations of agents $\mathcal{L} = (l_0, l_1, \ldots, l_{k-1})$ implies that each agent a_i stays at node v_{l_i}. We define C as a set of all configurations in an agent system. In initial configuration $c_0 \in C$, each agent holds the same initial state $s_{initial}$ and the whiteboard of each node is empty. This means that the initial configuration depends only on the locations of agents. In the initial configuration, multiple agents may stay at the same node.

Let A_i be an arbitrary non-empty set of agents. When configuration c_i changes to c_{i+1} by actions of every agent in A_i, we denote the transition by $c_i \xrightarrow{A_i} c_{i+1}$. When $a_j \in A_i$ moves to the next node or changes some states (of its own or the whiteboard), we say agent a_j takes one step. If multiple agents at the same node are included in A_i, the agents take steps in an arbitrary order. If sequence of configurations $E = c_0, c_1, \ldots$ satisfies $c_i \xrightarrow{A_i} c_{i+1} (i \geq 0)$, E is called an *execution* starting from c_0. Execution E is infinite, or ends in final configuration C_{final} where no agent can take a step. When $A_i = A$ holds for any i, all agents perform simultaneously. This model is called the *synchronous* model. Otherwise, the model is called the *asynchronous* model.

Rendezvous Problem. In this section, we formally define the rendezvous problem. First, we give the definition of *the rendezvous problem with (terminal) detection.* A *halt* state is defined as the state in which the agent never takes a step. That is, if an agent transits to a halt state, it can detect its termination. The traditional rendezvous problem requires agents to stop with such terminal detection in the final configuration. We call this traditional rendezvous problem the rendezvous problem with detection. We assume without loss of generality that agents have a unique halt state.

Definition 1. *Execution E solves the rendezvous problem with detection if the following conditions hold: 1) Execution E is finite, and 2) in the final configuration, all agents meet at a single node and hold the halt state.*

We also define *the rendezvous problem without detection.* A *suspending* state is defined as a state in which the agent never takes a step unless the whiteboard of the current node is updated by other agents. The rendezvous problem without detection allows agents to stop at suspending states. An agent at a suspending state cannot detect its termination because other agents may wake up the agent by updating the whiteboard state of the node.

Definition 2. *Execution E solves the rendezvous problem without detection if the following conditions hold: 1) Execution E is finite, and 2) in the final configuration, all agents meet at a single node and hold suspending states.*

Randomized algorithms for the rendezvous problem are defined as follows.

Definition 3. *A randomized algorithm solves, with probability p, the rendezvous problem with detection (resp., without detection) if the following condition holds: From any initial configuration c_0, executions which solve the rendezvous problem with detection (resp., without detection) occur with probability at least p.*

3 Impossibility Results

3.1 Impossibility of Rendezvous with Detection

In this section, we discuss impossibility of randomized rendezvous in anonymous unidirectional rings. We first consider the rendezvous problem with detection and prove that there exists no randomized algorithm that solves the problem. Note that the impossibility holds even for the synchronous model.

Theorem 1. *For any p ($0 < p \leq 1$), there exists no randomized algorithm that solves, with probability p, the rendezvous problem with detection in the synchronous model.*

We prove Theorem 1 by contradiction. We assume that an algorithm solves, with probability p, the rendezvous problem with detection.

We consider ring R that consists of n nodes $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and k agents $A = \{a_0, a_1, \ldots, a_{k-1}\}$. From the hypothesis, there exists an execution E that solves the rendezvous problem with detection. We define $T(E)$ as the

length of E and denote $E = c_0, c_1, \ldots, c_{T(E)}$. Note that every agent holds the halt state in the final configuration $c_{T(E)}$.

Next, we consider a larger ring R'. Let q be the minimum integer that satisfies $qn \geq T(E)$. We consider a fragment of R' consisting of $n' = 2qn + n$ consecutive nodes, and we call it *segment* S. We denote nodes in segment S by $v'_0, v'_1, \ldots, v'_{n'-1}$. We consider $k' = kq + k$ agents in the segment S. These agents are represented by $A_S = \{a'_0, a'_1, \ldots, a'_{k'-1}\}$. The initial location $(\ell'_0, \ell'_1, \ldots, \ell'_{k'-1})$ for the k' agents in segment S is defined from the initial location $(\ell_0, \ell_1, \ldots, \ell_{k-1})$ for k agents in ring R as follows:

$$\ell'_i = \ell_{i \bmod k} + n \cdot \lfloor i/k \rfloor.$$

That is, the initial positions of R are repeated from v'_0 to v'_{qn+n-1}, and there exist no agents from v'_{qn+n} to $v'_{2qn+n-1}$ (see Figure 1). For each node v'_j in R', we define $C_v(v'_j) = v_{j \bmod n}$ as the corresponding node of v'_j in R.

In the following, we first show that, with probability depending on n' and k', some agents in the segment S transit to the halt state without moving out of S. Then, we show that the algorithm fails to achieve rendezvous with probability $1 - p$ or more when the ring R' contains sufficiently many fragments with the same configuration as S. This contradicts the assumption.

First, we show that some agents in segment S perform in the same way as agents in R with non-zero probability. We define the *local configuration* of node v as the 2-tuple that consists of the state of v and the states of all agents at v.

Lemma 1. *Let us consider execution $E' = c'_0, c'_1, \ldots, c'_{T(E)}, \ldots$ for ring R'. We define $V'_t = \{v'_t, v'_{t+1}, \ldots, v'_{qn+n-1}\}$. Then, with non-zero probability, configuration c'_t meets the following condition:*

- *For each node $v'_j \in V'_t$, the local configuration of v'_j in c'_t is the same as that of $C_v(v'_j)$ in c_t.*

Proof. We prove Lemma 1 by induction on t. In the case of $t = 0$, Lemma 1 holds from the definition of R'. Next, we show that, when Lemma 1 holds for t ($t < T(E)$), Lemma 1 holds for $t + 1$.

From the induction hypothesis, with non-zero probability, the local configuration of v'_j in c'_t is the same as that of $C_v(v'_j)$ in c_t. Then, we assume that agents at the node $v'_j \in V'_t$ in c'_t generate the same random numbers as those agents at the node $C_v(v'_j)$ generate in c_t. The probability that such execution occurs is obviously more than 0.

Remind that, for each $v'_j \in V'_{t+1}$, the local configurations of v'_{j-1} and v'_j in c'_t are the same as those of $C_v(v'_{j-1})$ and $C_v(v'_j)$ in c_t. Consequently, agents at v'_{j-1} and v'_j behave in c'_t in the same way as those at $C_v(v'_{j-1})$ and $C_v(v'_j)$ in c_t. Since only agents at nodes v'_{j-1} and v'_j can change the local configuration of v'_j in unidirectional rings, the local configuration of v'_j in c'_{t+1} is the same as that of $C_v(v'_j)$ in c_{t+1}. Therefore, we have Lemma 1. □

From Lemma 1, in the segment S for R', the configuration $c'_{T(E)}$ satisfies the condition of Lemma 1 with some non-zero probability $p' > 0$. Then, in $c'_{T(E)}$, the

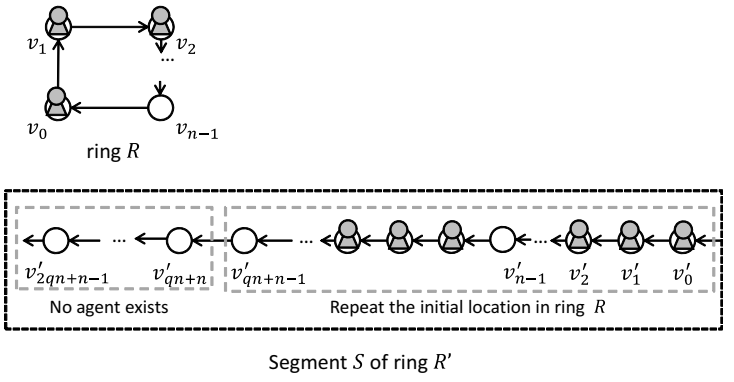


Fig. 1. The initial configuration of R and R'

local configuration of each node in $V^* = \{v'_{qn}, v'_{qn+1}, \ldots, v'_{qn+n-1}\} \subseteq V'_{T(E)}$ is the same as that of the corresponding node in $c_{T(E)}$. Note that the set of nodes corresponding to nodes in V^* is equal to V, and every agent stops with the halt state in configuration $c_{T(E)}$. Hence agents in V^* also stop with the halt state in configuration $c'_{T(E)}$. Another important point is that the probability p' in the above depends only on n' and k' (and does not depend on the size of R' or the total number of agents in R'). This obviously holds since the proof of Lemma 1 considers the execution of only segment S.

In ring R' of length xn' or more, we consider the initial configuration such that there exist x disjoint segments each of which has the same configuration as the above described one of segment S. In segment S, since there exists no agent from v'_{qn+n} to $v'_{2qn+n-1}$ initially, no agent moves out of segment S before configuration $c'_{T(E)}$. So, each of the x segments does not influence the other one, and the above execution in segment S occurs independently. Therefore, the probability P that agents stop with the halt state in at least two segments is $P = 1 - (1-p')^x - x(1-p')^{x-1}p'$. If agents stop with the halt state on the two segments, the agents cannot meet at a single node. Hence such execution does not solve the rendezvous problem with detection. That is, the probability that this algorithm solves the rendezvous problem with detection is at most $1 - P$. When the size of ring R' and x is sufficiently large, $1 - P < p$ holds. This contradicts the claim that, for the ring R', this algorithm solves, with probability p, the rendezvous problem with detection. Therefore, we have Theorem 1. □

3.2 Impossibility of Rendezvous without Detection

Similarly we have the following theorem for the rendezvous problem without detection. Note that the theorem holds even for the synchronous model.

Theorem 2. *There exists no randomized algorithm that solves, with probability* 1*, the rendezvous problem without detection.*

Proof. Due to limitation of space, we give only a brief sketch of the proof. We can prove the theorem similarly to Theorem 1. For contradiction, we assume that algorithm A solves the problem with probability 1. Considering a n-node ring R with k agents, all the agents meet at a single node and stop with the suspending states in R.

Next consider a $2n$-node ring R' with $2k$ agents. The initial positions of agents in R' are repeated twice in R. Then, all the agents in R' can behave in the same way as those in R. Consequently a half of the agents meet at a single node and the other half of the agents meet at another node. All agents stop with the suspending states and thus A cannot solve the rendezvous problem without detection. □

4 A Randomized Algorithm for Rendezvous without Detection

In this section, we propose a randomized algorithm that solves, with any given probability $p(0 < p < 1)$, the rendezvous problem without detection. The idea of the algorithm is to assign a random ID to each node so that agents can identify a unique rendezvous node by the IDs.

In the following, we present the algorithm. The probability p to solve the rendezvous problem is given as an input to the algorithm. Let p_1 and p_2 be any positive real numbers that satisfy $p = p_1 p_2$. Throughout the section, when an agent writes a random bit to a variable, it chooses one or zero uniform-randomly.

First, we show the overview of the algorithm. The algorithm consists of four phases. In phase 1, each agent moves in the ring and appends a random bit to the whiteboard every time it visits a node. So the bit sequence assigned to a node is extended by one bit every time the node is visited by an agent. We consider the bit sequence as a label of the node. Based on the random bits assigned to each node, the agent distinguishes nodes and guesses the number of nodes, say n'. In phase 2, the agent moves on n' nodes and gives random IDs to the nodes. In phase 3, the agent moves on n' nodes and write the value n' to the whiteboards. After that the agent identifies the rendezvous node and moves to the node. Note that, if n' is correctly guessed and is equal to n, every node has an ID and knows the number of nodes after phases 2 and 3. At the end of phase 3, the agent becomes a suspending state. However, when n' is not correctly guessed and is different from n, the agent may prematurely stop moving and become a suspending state. Even in this case, if another agent guesses the correct number n, it writes the value n to the whiteboard of every node in phase 3. Consequently the agent at the suspending state can notice its wrong guess on the number of nodes and resume its behavior. This is phase 4, and in this phase the agent moves throughout the ring, memorizes the IDs, and then moves to the rendezvous node. Note that, if at least one agent succeeds to guess the correct number of nodes, every agent can know the correct number of nodes and IDs of all nodes. This fact is very useful to guarantee that all the agents can meet at a single node with probability p or more.

In the rest of this section, we give the details of each phase. The pseudocode is given in Algorithms 1 and 2. In the pseudocode, we denote by node v_j the node where the agent currently stays.

Phase 1. The goal of phase 1 is that each agent guesses the number of nodes. The fundamental idea of estimating the number of nodes is to assign a random label to each node and to consider that the agent circulates the ring twice when it finds a repeated sequence of the labels. Agent a_i uses variables $memory1$, t, n' on agent a_i and $wb1_j$ on node v_j. Variable $wb1_j$ stores a random label of node v_j, and $memory1$ is used to store the sequence of node labels the agent visited. Variable t stores the number of moves in phase 1. Variable n' stores the number of nodes that a_i guesses. We call n' the temporary number of nodes.

Agent a_i executes the following procedure in each step. Assume that a_i stays at node v_j.

- Agent a_i appends one random bit to $wb1_j$ on v_j, that is, a_i writes one random bit on $wb1_j[x+1]$ when variables $wb1_j[1..x]$ are already written.
- Agent a_i assigns $wb1_j$ (the random bits) to $memory1[t]$, where t is the number of moves the agent made.
- Agent a_i guesses the number of nodes from $memory1$ (see details in the below) and assigns it to n'.
- Agent a_i moves to the next node.

This step is repeated until a_i guesses some temporal number of nodes n' and the number of moves is at least $3n' + \log \frac{1}{1-p_1}$. As we show later, this number of moves guarantees that agent a_i guesses the correct number of nodes with probability p_1.

Here we explain the way to guess the number of nodes from $memory1$ (Function Guess() in Algorithm 2). Because agent a_i moves repeatedly, it eventually circulates the ring twice. Then, each value in the first half of $memory1$ is a prefix of the value of the corresponding entry in the latter half of $memory1$. For this reason, the agent guesses the number of nodes when each value in the first half of $memory1$ appears as a prefix of the corresponding value in the latter half of $memory1$. More concretely the agent guesses that the number of nodes is d where $d (1 \leq d \leq \lfloor t/2 \rfloor)$ is the maximum value satisfying for any x $(0 \leq x \leq t-d)$ and any y $(0 \leq y \leq |memory1[x]| - 1)$, $memory1[x][y] = memory1[x+d][y]$ and $|memory1[x]| \neq |memory1[x+d]|$ hold (see Figure 2). The second inequality means that, if the number of nodes is d, additional random bits should be written to the whiteboard in the second visit. If there does not exist such d, the agent continues phase 1 to guess the number of nodes and assigns zero to n'.

Phase 2. In phase 2, agent a_i gives a random ID to each node. The variable $wb2_j$ on node v_j is used to store the ID that an agent generates randomly. Agent a_i moves over the ring and stores all IDs in $memory2$. In more details, agent a_i executes the following procedure in each step. Assume that a_i stays at node v_j.

- Agent a_i generates a random ID of length $\lceil \frac{2}{n'} \log \frac{n'}{1-p_2} \rceil$ and writes the ID to $wb2_j$ unless $wb2_j$ already stores an ID.

Algorithm 1. Proposed Algorithm

input
float $p \in (0,1)$ //p satisfies $p = p_1 p_2$
Variables in Agent a_i
int $t = 0$ // the number of moves in each phase
boolean[ ][ ] $memory1$,$memory2$
int $v_{min} = 0$
int $n' = 0$ // the number of nodes that agent a_i guesses
Variables in Node v_j
boolean[ ] $wb1_j$,$wb2_j$
int $size_j = 0$
Main Routine of Agent a_i
Phase1
1: **while** ($n' = 0$ or $t < 3n' + \log \frac{1}{1-p_1}$) **do**
2: a_i appends one random bit to $wb1_j$ // When variables $wb1_j[1..x]$ are already written, a_i writes one random bit on $wb1_j[x+1]$.
3: a_i assigns $wb1_j$ to $memory1[t]$
4: $n' =$ Guess()
5: $t++$
6: a_i moves to the next node
7: **end while**

Phase2
8: **for** $t = 0$ to $n' - 1$ **do**
9: a_i generates a random ID of length $\lceil \frac{2}{n'} \log \frac{n'}{1-p_2} \rceil$ and writes the ID to $wb2_j$ unless $wb2_j$ already stores an ID.
10: a_i assigns $wb2_j$ to $memory2[t]$
11: a_i moves to the next node
12: **end for**

Phase3
13: **for** $t = 0$ to $n' - 1$ **do**
14: **if** $size_j < n'$ **then**
15: $size_j = n'$
16: **end if**
17: a_i moves to the next node
18: **end for**
19: $v_{min} = Rendezvousnode()$
20: **for** $\ell = 0$ to $v_{min} - 1$ **do**
21: a_i moves to the next node
22: **end for**
23: a_i stops with the suspending state

- Agent a_i assigns $wb2_j$ to $memory2[t2]$, where $t2$ is the number of moves that a_i made in phase 2.
- Agent a_i moves to the next node.

Algorithm 2. Proposed Algorithm

Phase4

while 1 **do**
  if $n' < size_j$ **then**
    $n' = size_j$
    for $t = 0$ to $n' - 1$ **do**
      a_i assigns $wb2_j$ to $memory2[t]$
      a_i moves to the next node
    end for
    $v_{min} = Rendezvousnode()$
    for $t = 0$ to $v_{min} - 1$ **do**
      a_i moves to the next node
    end for
    a_i stops with the suspending state
  end if
end while

int Guess()

D = the set of d $(1 \leq d \leq \lfloor t/2 \rfloor)$ such that $\forall x$ $(0 \leq x \leq t - d + 1)$ $\forall y$ $(0 \leq y \leq |memory1[x]| - 1)$ $memory1[x][y] = memory1[x + d][y]$ and $|memory1[x]| \neq |memory1[x + d]|$.

If $D \neq \emptyset$, return $\min(D)$. If $D = \emptyset$, return 0.

int Rendezvousnode()

Let seq_x $(0 \leq x \leq n' - 1)$ be the cyclically shifted sequence $(memory2[x], \ldots, memory2[n'-1], memory2[0], \ldots, memory2[x-1])$ of $memory2$.

Let x^* be x such that seq_x is the lexicographically minimum among seq_0 to $seq_{n'-1}$.

return the distance from the current node to node v_j such that $wb2_j = memory2[x^*]$. Note that the agent computes the distance based on its temporary number of nodes.

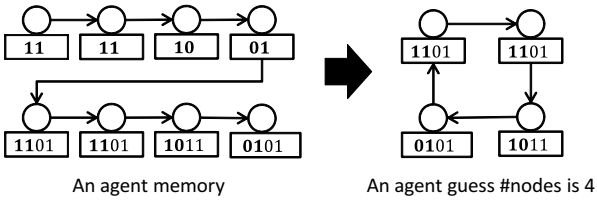


Fig. 2. The way that an agent guesses the number of nodes

Agent a_i executes the above step n' times, and consequently every node is assigned an ID if $n' = n$ holds. Then, as we show later, if $n' = n$ holds, the above length of ID guarantees that all agents can choose the identical node as the rendezvous node with probability p_2. Since $n' = n$ holds with probability p_1, with probability $p_1 p_2 = p$, all agents can choose the identical rendezvous node.

Phase 3. In phase 3, agent a_i writes the temporal number of nodes n' to $size_j$ of each node v_j and then moves to the identical node where all agents meet. The process is easily realized by moving n' times and assigning n' to $size_j$ on each node v_j. When $size_j$ already stores a value larger than n', agent a_i does not overwrite $size_j$ because n' is not equal to n. If $n' = n$ holds, every agent can get the correct number of nodes. As we explain in phase 4, this process is used to wake up agents that are at the suspending states with keeping wrong (or smaller) numbers of nodes.

Now we show how all the agents can choose the identical node. Agent a_i chooses the node based on $memory2$. To be concrete, a_i chooses it in the following way (Function Rendezvousnode() in Algorithm 2):

- Let seq_x $(0 \leq x \leq n'-1)$ be the cyclically shifted sequence $(memory2[x], \ldots, memory2[n'-1], memory2[0], \ldots, memory2[x-1])$ of $memory2$.
- Let x^* be x such that seq_x is the lexicographically minimum among seq_0 to $seq_{n'-1}$.
- Node v_j corresponding to the first node in seq_{x^*} (or $wb2_{(j+y) \bmod n'} = memory2[(x^* + y) \bmod n']$ for $y(0 \leq y \leq n'-1)$) is the identical node where all agents meet.

Note that, if the contents of $memory2$ (i.e., the sequence of node IDs) are periodic, there exists more than one x such that seq_x is the lexicographically minimum. In this case, agents fail to choose the identical node and cannot meet at a single node. However, we can show that the probability of such failure is sufficiently small. If the contents of $memory2$ are not periodic, all the agents can choose the identical node and succeed to rendezvous at the node. After a_i reaches the node, it becomes the suspending state.

Phase 4. Phase 4 specifies the behavior after a_i stops with the suspending state. When a_i is at the suspending state, it repeatedly checks $size_j$ on its current node v_j. If another agent updates $size_j$ to a value larger than n', agent a_i recognizes that it failed to estimate n correctly. Then, a_i wakes up and moves to the node to meet other agents. The behavior of a_i is as follows:

- Agent a_i assigns $size_j$ to n'.
- Agent a_i moves n' times, and it memorizes $wb2_j$ on each node v_j from $memory2[0]$ to $memory2[n'-1]$.
- The agent chooses the node to meet other agents by the same way as that in phase 3, and then moves to the node.
- Agent a_i stops with the suspending state.

Note that, if an agent guesses the correct number of nodes, it writes n' to whiteboards of all nodes. Then, in phase 4, other agents can know the correct number of nodes and compute the identical node.

We have the following theorem about the algorithm. Due to limitation of space, the proof is omitted.

Theorem 3. *The algorithm in Algorithms 1 and 2 solves, with probability p, the rendezvous problem without detection. In the algorithm each agent moves $O(kn)$ times and has $O(kn)$ memory space and each node has $O(k+\log n)$ memory space.*

5 Conclusion

In this paper, we considered the randomized rendezvous of mobile agents in anonymous unidirectional ring network under the assumption that each mobile agent knows neither the number of nodes nor the number of mobile agents. We showed the impossibility result for the problem with detection, and the possibility and impossibility results (dependent on the success probability) for the problem without detection. In the future work, we will propose an algorithm for arbitrary networks through the approach that each agent constructs a labeled network from an anonymous arbitrary network.

References

1. Alpern, S., Baston, V.J., Essegaier, S.: Rendezvous search on a graph. Journal of Applied Probability 36(1), 223–231 (1999)
2. Baba, D., Izumi, T., Ooshita, F., Kakugawa, H., Masuzawa, T.: Space-Optimal Rendezvous of Mobile Agents in Asynchronous Trees. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO 2010. LNCS, vol. 6058, pp. 86–100. Springer, Heidelberg (2010)
3. Barriere, L., Flocchini, P., Fraigniaud, P., Santoro, N.: Rendezvous and election of mobile agents: impact of sense of direction. Theory of Computing Systems 40(2), 143–162 (2007)
4. Chalopin, J., Das, S., Widmayer, P.: Rendezvous of Mobile Agents in Directed Graphs. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 282–296. Springer, Heidelberg (2010)
5. Das, S., Mihalák, M., Šrámek, R., Vicari, E., Widmayer, P.: Rendezvous of Mobile Agents When Tokens Fail Anytime. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 463–480. Springer, Heidelberg (2008)
6. Dieudonné, Y., Pelc, A.: Deterministic gathering of anonymous agents in arbitrary networks. Arxiv preprint arXiv:1111.0321 (2011)
7. Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: Proc. of SODA, pp. 585–594 (2007)
8. Kranakis, E., Krizanc, D.: An Algorithmic Theory of Mobile Agents. In: Montanari, U., Sannella, D., Bruni, R. (eds.) TGC 2006. LNCS, vol. 4661, pp. 86–97. Springer, Heidelberg (2007)
9. 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)
10. Kranakis, E., Krizanc, D., Markou, E.: The mobile agent rendezvous problem in the ring. Synthesis Lectures on Distributed Computing Theory, Lecture # 1. Morgan & Claypool Publishers (2010)
11. Kranakis, E., Krizanc, D., Morin, P.: Randomized Rendez-Vous with Limited Memory. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 605–616. Springer, Heidelberg (2008)
12. Sudo, Y., Baba, D., Nakamura, J., Ooshita, F., Kakugawa, H., Masuzawa, T.: An agent exploration in unknown undirected graphs with whiteboards. In: Proc. of WRAS, p. 8 (2010)

Getting Close without Touching*

Linda Pagli, Giuseppe Prencipe, and Giovanni Viglietta

Dipartimento di Informatica, Università di Pisa
{pagli,prencipe,viglietta}@di.unipi.it

Abstract. In this paper we study the Near-Gathering problem for a set of asynchronous, anonymous, oblivious and autonomous mobile robots with limited visibility moving in Look-Compute-Move (LCM) cycles: In this problem, the robots have to get close enough to each other, so that every robot can see all the others, without touching (i.e., colliding) with any other robot. The importance of this problem might not be clear at a first sight: Solving the Near-Gathering problem, it is possible to overcome the limitations of having robots with limited visibility, and it is therefore possible to exploit all the studies (the majority, actually) done on this topic, in the unlimited visibility setting. In fact, after the robots get close enough, they are able to see all the robots in the system, a scenario similar to the one where the robots have unlimited visibility. Here, we present a collision-free algorithm for the Near-Gathering problem, the first to our knowledge, that allows a set of autonomous mobile robots to nearly gather within finite time. The collision-free feature of our solution is crucial in order to combine it with an unlimited visibility protocol. In fact, the majority of the algorithms that can be found on the topic assume that all robots occupy distinct positions at the beginning. Hence, only providing a collision-free Near-Gathering algorithm, as the one presented here, is it possible to successfully combine it with an unlimited visibility protocol, hence overcoming the natural limitations of the limited visibility scenario. In our model, distances are induced by the infinity norm. A discussion on how to extend our algorithm to models with different distance functions, including the usual Euclidean distance, is also presented.

1 Introduction

Consider a distributed system whose entities are a set of *robots* or *agents* that can freely move on a two-dimensional plane, operating in *Look-Compute-Move* (LCM) cycles. During a cycle, a robot takes the snapshot of the position of the other robots (*Look*); executes the protocol, the same for all robots, using the snapshot as an input (*Compute*); and moves towards the computed destination, if any (*Move*). After each cycle, a robot may be inactive for some time. With respect to the LCM cycles, the most common models used in these studies are the *fully synchronous* (FSYNC), the *semi-synchronous* (SSYNC), and

* This work has been partially supported by MIUR of Italy under projects MadWeb and AlgoDEEP prot. 2008TFBWL4.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 315–326, 2012.

the *asynchronous* (ASYNC). In the *asynchronous* (ASYNC) model, each robot acts independently from the others and the duration of each cycle is finite but unpredictable; thus, there is no common notion of time, and robots can compute and move based on *obsolete* observations. In contrast, in the *fully synchronous* (FSYNC) model, there is a common notion of time, and robots execute their cycles synchronously. In particular, time is assumed to be discrete, and at each time instant *all* robots are activated, obtain the same snapshot, compute and move towards the computed destination; thus, no computation or move can be made based on obsolete observations. The last model, the *semi-synchronous* (SSYNC), is like FSYNC where, however, not all robots are necessarily activated at each time instant.

In the last few years, the study of the computational capabilities of such a system has gained much attention, and the main goal of the research efforts has been to understand the relationships between the capabilities of the robots and their power to solve common tasks. The main capabilities of the robots that, to our knowledge, have been studied so far in this distributed setting are *visibility*, *memory*, *orientation*, and *direct communication*. With respect to visibility, the robots can either have *unlimited visibility*, by sensing the positions of *all* other robots, or have *limited visibility*, by sensing just a portion of the plane, in particular up to a given distance V [1,8]. With respect to memory, the robots can either be *oblivious*, by having access only to the information sensed or computed during the current cycle (e.g., [14]), or *non-oblivious*, by having the capability of storing the information sensed or computed since the beginning of the computation (e.g., [2,15,16]). With respect to orientation, the two extreme settings studied are the one where the robots have *total agreement*, and agree on the orientation and direction of their local coordinate systems (i.e., they agree on a *compass*), e.g., [9], and the one where the robots have *no agreement* on their local coordinate axes, e.g., [15,16]; in the literature, there are studies that tackle also the scenarios in between; for instance, when the robots agree on the direction and orientation just of the y coordinate, or there is agreement just on the chirality of the coordinate system, e.g., [6]. With respect to direct communication, the direction so far has been towards the use of external signals or lights to enhance the capabilities of mobile, first suggested in [12], and also referenced in [7], which provided the earliest indication that incorporating in the robot model some simple means of signalling might positively affect the power of the team. Recently, a study that tackles more systematically this particular capability has been presented in [3].

In this paper, we solve the Near-Gathering problem: The robots are required to get close enough to each other, without touching or colliding during their movements. Here, the team of robots under study executes the cycles according to the ASYNC model, the robots are oblivious and have limited visibility. The importance of this problem might not be clear at a first sight: With a solution to the Near-Gathering problem it would be possible to overcome the limitations of having robots with limited visibility, and it would be possible to exploit all the studies (the majority, actually) done in the unlimited visibility

setting. In fact, after the robots get close enough, they are able to see all the robots in the system, a scenario similar to the one where the robots have unlimited visibility. Since most of the solutions to the unlimited visibility case assume a starting configuration where no two robots *touch* (i.e., they do not share the same position in the plane), it is of crucial importance to ensure that no collision occurs during the near gathering.

A problem close to Near-Gathering is the *gathering* problem, where the robots have to meet, within finite time, in a point of the plane not agreed in advance. This problem has been studied in the literature in all models; in particular, a study in SSYNC with limited visibility has been presented in [1]: Actually, this solution could be easily modified to solve also the Near-Gathering problem, just imposing a termination condition; however, it has been shown that this solution does not work in ASYNC [13]. Another solution for the limited visibility case is in [14], where the coordinate systems are assumed to be consistent only after a period of instability (i.e., the robots agree on the coordinate system only after an arbitrary long period); however, also this solution is designed for the SSYNC model. In [10] a convergence protocol that works with a very limited form of asynchrony (called 1-bounded asynchrony) has been presented. In the asynchronous model, the only solution to the gathering problem with robots having limited visibility has been presented in [8]: This protocol, however, is not collision-free; hence, it cannot be used to solve our problem. We note that, as in the protocol in [8], we also assume that the robots have total agreement. Also, we remark that, since the algorithm presented here is for the ASYNC model, it solves the problem also in the SSYNC and FSYNC models.

As stated above, solutions to problems studied in the unlimited visibility setting can be potentially used to solve the same problems in the limited visibility setting, by exploiting the Near-Gathering protocol presented in this paper. Among these, we can cite for instance the *Arbitrary Pattern Formation Problem* [9,6,15,16], or the *Uniform Circle Formation* (e.g., [4,5]).

The organization of the paper is as follows: In Section 2 the formal definition of the robot model is presented; in Section 3 the collision-free algorithm that solves the Near-Gathering problem is presented; in Section 4 the correctness of the protocol is shown. Due to space constraints, the proofs will be omitted, and we thoroughly discuss only the scenario in which distances are induced by the infinity norm (the full version of the paper can be found in [11]). However, some extensions of our algorithm to models with different distance functions, including the usual Euclidean distance, are also briefly discussed in Section 5.

2 The Model

The system is composed of a team of mobile entities, called *robots*, each modeled as a computational unit provided with its own local memory and capable of performing local computations. The robots are (viewed as) points in the plane. Let $r(t)$ denote the absolute position of robot r at time t (i.e., with respect to an absolute reference frame); also, we will denote by $r(t).x$ and $r(t).y$ the abscissa

and the ordinate value of position $r(t)$, respectively. When no ambiguity arises, we shall omit the temporal indication; also, the *configuration of the robots at time t* is the set of robots' positions at time t.

Each robot has its own local coordinate system, and we assume that the local coordinate systems of the robots are consistent with each other: In other words, they agree on where the North, South, East and West are. A robot is endowed with sensorial capabilities and it observes the world by activating its sensors, which return a snapshot of the positions of all other robots with respect to its local coordinate system. The visibility radius of the robots is limited: Robots can sense only points in the plane within distance V. This setting, referred in the literature as *limited visibility*, is understandably more difficult; for example, a robot with limited visibility might not even know the total number of robots nor where they are located if outside its radius of visibility. Also, combined with the asynchronous behavior of the robots, introduces a higher level of difficulty in the design of collision-free protocols. For instance, in the example depicted in Figure 1.a, robot s, in transit towards its destination, is seen by r; however, s is not aware of r's existence and, if it starts the next cycle before r starts moving, s will continue to be unaware of r; hence, since r does not see s when s starts its movement, it must take care of the "potential" arrival of s when computing its destination.

All robots are identical: They are indistinguishable from their appearance and they execute the same protocol. Robots are autonomous, without a central control. Robots are silent, in the sense that they have no means of direct communication (e.g., radio, infrared) of information to other robots. Each robot is endowed with motorial capabilities, and can move freely in the plane. A move may end before the robot reaches its destination, e.g., because of limits to its motion energy. The distance traveled in a move is neither infinite nor infinitesimally small. More precisely, there exists a constant $\delta > 0$ such that, if the destination point is closer than δ, the robot will reach it; otherwise, it will move towards it of at least δ. Note that, without this assumption, an adversary would make it impossible for any robot to ever reach its destination, following a classical Zenonian argument. The quantity δ might not be known to the robots.

The robots do not have persistent memory, that is, memory whose content is preserved from one cycle to the next; they are said to be *oblivious*. The only available memory they have is used to store local variables needed to execute the algorithm at each cycle.

At any point in time, a robot is either *active* or *inactive*. When *active*, a robot r executes a *Look-Compute-Move* (LCM) cycle performing the following three operations, each in a different state: (i) **Look:** The robot observes the world by activating its sensor, which returns a snapshot of the positions of all robots within its radius of visibility with respect to its own coordinate system (since robots are viewed as points, their positions in the plane are just the set of their coordinates); (ii) **Compute:** The robot executes its algorithm, using the snapshot as input. The result of the computation is a destination point; (iii) **Move:** The robot moves towards the computed destination; if the destination

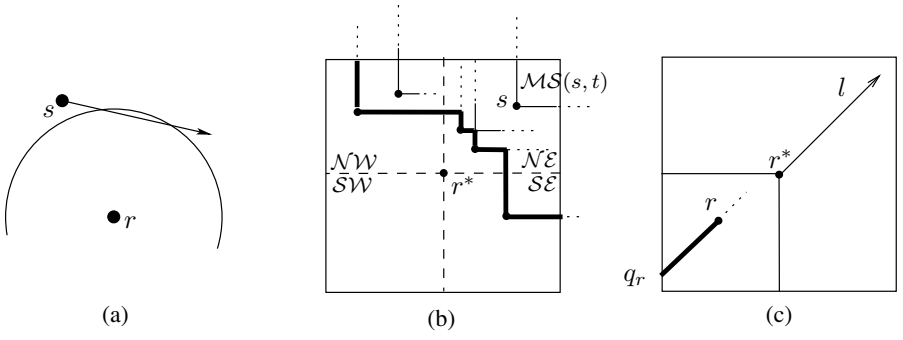


Fig. 1. (a) When s starts moving (the left end of the arrow), r and s do not see each other. While s is moving, r *Look*s and sees s; however, s is still unaware of r. After s passes the area of visibility of r, it is still unaware of r. (b) The area above and to the right of s defines the *Move Space* of s. The fat line is the *Contour* of r^*. (c) Computation of the length of the movement in the algorithm.

is the current location, the robot stays still (performs a *null movement*). When *inactive*, a robot is idle. All robots are initially inactive. The amount of time to complete a cycle is assumed to be finite, and the *Look* is assumed to be instantaneous. We will denote by $\mathbb{W}(t)$, $\mathbb{L}(t)$, $\mathbb{C}(t)$, $\mathbb{M}(t)$ the sets of robots that are, respectively, inactive, in a *Look* phase, in a *Compute* phase and in a *Move* phase at time t.

In the following, we will assume that all distances are induced by the *infinity norm*: $\|p\|_\infty = \max\{p.x, p.y\}$. Different distance functions, including the usual Euclidean distance, will be briefly discussed in Section 5.

2.1 Notation

We will denote by $\mathcal{R} = \{r_1, \ldots, r_n\}$ the set of robots in the system. First note that, in order to achieve explicit termination, it is necessary that all robots share the knowledge of n. In Section 4.5 we will show how to overcome this by making use of visible bits [3].

We will denote by $G(t) = (N, E(t))$ the *distance graph* at time $t \geq 0$, where N is the set of the input robots and, for any two distinct robots r and s, $(r, s) \in E(t)$ iff $0 \leq \|r(t) - s(t)\|_\infty \leq V$. In [8] it was proved that the initial distance graph $G(0)$ must be connected for the gathering problem to be solvable; the same result clearly holds also for the Near-Gathering problem. Thus, in the following we will always assume that $G(0)$ is connected.

Let r be a robot, and let us divide its visible area into four quadrants, denoted by $\mathcal{NW}(r)$, $\mathcal{NE}(r)$, $\mathcal{SE}(r)$, and $\mathcal{SW}(r)$ (see the example depicted in Figure 1.b). For technical reasons, the vertical and the horizontal segment of length V starting from r and going South and West, respectively (including the location of r itself), are part of $\mathcal{SW}(r)$; the vertical (resp. horizontal) segment of length V passing

through r and going North (resp. East) is part of $\mathcal{NW}(r)$ (resp. $\mathcal{SE}(r)$). When not necessary, the reference to r will be dropped. Similarly, a reference to time may be added.

Next, we define the *Move Space* of a robot (refer to the example depicted in Figure 1.b):

Definition 1 (Move Space). *The* Move Space *of a robot r at time t, denoted by $\mathcal{MS}(r,t)$, is the set $\{(x',y') \in \mathbb{R}^2 \mid x' \geq r(t).x \wedge y' \geq r(t).y\}$.*

Based on the previous definition, we introduce the *Contour* of a robot (refer again to Figure 1.b):

Definition 2 (Contour). *The* Contour *of a robot r at time t, denoted by $\mathcal{CT}(r,t)$, is the boundary of the set $\bigcup_s \mathcal{MS}(s,t)$, where s ranges through all the robots in $\mathcal{NW}(r,t) \cup \mathcal{NE}(r,t) \cup \mathcal{SE}(r,t)$.*

We will call a *peak* of the contour any convex corner of $\mathcal{CT}(r)$; the concave corners will be called *valleys*. An easy property of $\mathcal{CT}(r,t)$ is stated in the following

Observation 1. *If there are robots in both $\mathcal{NW}(r)$ and in $\mathcal{SE}(r)$, and no robot in $\mathcal{NE}(r)$, then $\mathcal{CT}(r)$ has exactly one valley in $\mathcal{NE}(r)$.*

3 The Near-Gathering Problem and Its Solution

In the Near-Gathering problem, at the beginning a set of n robots is arbitrarily placed in the plane, on distinct positions such that $G(0)$ is connected: We will call this the *initial configuration*, denoted by $\mathcal{I}$. In finite time, the robots are required to move within distance ε from each other, for a given $0 < \varepsilon < V/4$: We will call this the *final configuration*, denoted by $\mathcal{F}$.

Our solution is reported in Figure 2. Informally, at each cycle, robot r^* first computes the direction of movement according to the following rules:

- If r^* can see robots only in $\mathcal{SW}$, then it will not move; that is, in this case the destination point is the point of coordinates $(0,0)$.
- If r^* can see robots only in $\mathcal{NW} \cup \mathcal{SW}$, then its direction of movement is given by the half-line l starting in r^* and going North.
- If r^* can see robots only in $\mathcal{SW} \cup \mathcal{SE}$, then its direction of movement is given by the half-line l starting in r^* and going East.
- Otherwise, the direction of movements of r is decided based on the shape of the Contour of r^*. In particular, if in $\mathcal{NE}$ there is at least a robot, the direction of movement is given by the half-line l starting from r^* and passing through robot in $\mathcal{NE}$ closest to r^*. Otherwise, there must be robots in both $\mathcal{NW}$ and $\mathcal{SE}$; in this case, the direction of movement is given by the half-line l starting from r^* and passing through the only valley in $\mathcal{CT}(r^*)$.

In order to establish the length of the movements along l, r^* checks two main factors: First, it must not enter the Move Space of any robot it can see (this contributes to guarantee collision avoidance); second, the new position must be

within distance $V/2$ from any of the robots it is currently seeing (this contributes to guarantee both collision avoidance and the connectedness of the initial distance graph). In order to ensure these two factors, first, for each $r \in \mathcal{NW} \cup \mathcal{NE} \cup \mathcal{SE}$, it computes the intersection p_r between l and $\mathcal{MS}(r)$ (notice that robots move only upward and rightward). Second, for each visible robot r, the intersection q_r between the visible area of r^* and the line parallel to l and passing through r is computed: The distance d_r between r and q_r is the maximum distance r^* is allowed to move in order to not lose visibility with r (assuming r does not move). Thus, if p is the point closest to r^* among the points in $\{p_r\} \cup \{d_r\}$, the destination point of r^* is the median point dp on the segment between r^* and p.

As we will prove in the following, a consequence of the computation of dp as described above is that the distance graph never gets disconnected; also, collisions are avoided. Termination is achieved using the knowledge of n that the robots are assumed to have. In fact, it is easy to see that, since the robots operate in a totally asynchronous environment, without knowledge of n, explicit termination would not be possible. In particular, in our solution, a robot terminates its execution as soon as it sees n robots at distance less than a given tolerance ε.

4 Correctness

In this section, we will prove that the Algorithm reported in Figure 2 correctly solves the Near-Gathering problem. In particular, the proof will be articulated in three parts: First, we will prove that the initial distance graph is preserved during the execution; second, we will prove that no collision occurs during the movements of the robots; finally, the correctness proof concludes by showing that the algorithm terminates.

4.1 Preliminary Definitions and Observations

Before presenting the correctness proof, we will introduce a few preliminary definitions and observations. First, by construction, it is easy to observe the following:

Observation 2. *Each robot can only move rightward and upward. Furthermore, the robots on the rightmost vertical axis never move right, and the robots on the topmost horizontal axis never move up.*

Observation 3. *During each cycle, a robot travels a distance of at most $V/2$.*

Definition 3 (First and Last). *Given a robot r, let $First(r,t) = \min\{t' > t | r \in \mathbb{L}(t')\}$ be the first time, after time t, at which r performs a Look operation. Also, let $Last(r,t) = \max\{t' \leq t | r \in \mathbb{L}(t')\}$ be the last time, from the beginning up to time t, at which r has performed a Look operation; if r has not performed a Look yet, then $Last(r,t) = 0$.*

State *Look*

Take the snapshot of the positions of the visible robots, which returns, for each robot $r \in \mathcal{R}$ within distance V, $\texttt{Pos}[\texttt{r}]$, the position in the plane of robot r (according to my coordinate system); (**Note**: I am robot r^*)

State *Compute*

Z_ε = Robots in $\texttt{Pos}[]$ within distance $\leq \varepsilon$;
If $|Z_\varepsilon| = n$ **Then Terminate**.
$l, p_1, \ldots, p_n, p'_1, \ldots, p'_n, b = \texttt{nil}$;
Let $\mathcal{NW}$, $\mathcal{NE}$, $\mathcal{SE}$, and $\mathcal{SW}$ be the quadrants of my visible area;
$\mathcal{CT}$ = Contour of the robots in $\mathcal{NW} \cup \mathcal{NE} \cup \mathcal{SE}$;
If I see robots only in $\mathcal{SW}$ **Then** $dp = (0, 0)$;
Else
  If I see robots only in $\mathcal{NW} \cup \mathcal{SW}$ **Then**
    l = Half-line from me going North;
  Else If I see robots only in $\mathcal{SE} \cup \mathcal{SW}$ **Then**
    l = Half-line from me going East;
  Else
    If There is at least one robot in $\mathcal{NE}$ **Then**
      l = Half-line from me to the closest robot in $\mathcal{NE}$;
    Else
      l = Half-line from me to the only valley of $\mathcal{CT}$ in $\mathcal{NE}$;
  For Each robot $r \in \mathcal{NW} \cup \mathcal{NE} \cup \mathcal{SE}$ **Do**
    p_r = Intersection between l and $\mathcal{MS}(r)$;
  For Each visible robot r **Do**
    l_r = Line parallel to l and passing through r;
    q_r = Lowest or leftmost intersection between l_r and my visible area;
    d_r = Distance between r and q_r;
    p'_r = Point on l at distance d_r from me;
  b = Point on l at distance V from me;
  p = Point closest to me among points in $\{p_r\} \cup \{p'_r\} \cup \{b\}$;
  dp = Median point on the segment between my position and p.

State *Move*

$\texttt{Move}(dp)$.

Fig. 2. The NEAR-GATHERING Protocol

Now, we define the *Destination Point* of a robot at a time t as follows:

Definition 4 (Destination Point). *Given a robots r, we define the* Destination Point $\texttt{DP}(r, t)$ *of r at time t as follows:*

- *If $r \in \mathbb{W}(t)$, then: if r is in its first cycle, then $\texttt{DP}(r, t) = r(0)$ (i.e., the starting position of r); otherwise, $\texttt{DP}(r, t)$ is the point p as computed in the last Compute state before t (in the previous cycle).*

- *If* $r \in \mathbb{L}(t)$*, then* $\texttt{DP}(r,t)$ *is the point* p *as computed in the next Compute state after* t *(in the current cycle).*
- *If* $r \in \mathbb{C}(t)$*, then* $\texttt{DP}(r,t)$ *is the point* p *as computed in the current Compute state.*
- *If* $r \in \mathbb{M}(t)$*, then* $\texttt{DP}(r,t)$ *is the point* p *as computed in the last Compute state before* t *(in the current cycle).*

From the previous definition, we can state the following:

Lemma 1. *Let* r *be a robot. During the time strictly between two consecutive Looks, the Destination Point of* r *does not change.*

4.2 Preservation of Mutual Awareness

We will now prove that the connectedness of the initial distance graph is preserved during the entire execution of the algorithm. We do so by first introducing the notion of *mutual awareness.*

Definition 5 (Mutual Awareness). *Two distinct robots* r *and* s *are* mutually aware *at time* t *iff both conditions hold:*

1. $\|r(t_r) - s(t_r)\|_\infty \leq V$*, with* $t_r = Last(r,t)$*, and*
2. $\|r(t_s) - s(t_s)\|_\infty \leq V$*, with* $t_s = Last(s,t)$*.*

Since initially all robots are inactive, then by definition of mutual awareness we have

Lemma 2. *All the pairs of robots that are within distance* V *from each other at time* $t = 0$ *are initially mutually aware.*

In the following lemma, we will prove that two robots that are mutually aware at the beginning of the computation keep the awareness during the execution.

Lemma 3. *If robots* r *and* s *are mutually aware at time* t*, they are mutually aware at any time* $t' > t$*.*

Based on the previous lemma, we can state the following

Corollary 1. *The connectedness of* $G(0)$ *is preserved during the execution of the algorithm.*

4.3 Collision Avoidance

In this section, we will prove that no collision occurs during the execution of the algorithm.

Lemma 4. *No collision ever occurs between any pair of robots during the execution of the algorithm.*

4.4 Termination

Let us call *Right* the vertical axis passing throught the righmost robot(s) in $\mathcal{I}$, and *Top* the horizontal axis passing throught the topmost robot(s) in $\mathcal{I}$; also, let f be the intersection point between *Right* and *Top*. By Observation 2, and by the algorithm, we can easily observe that

Observation 4. *If at any time t a robot is at position f, then it never moves from there.*

Next, we introduce a definition that will be useful to prove the convergence of the algorithm.

Definition 6 (Convergence Point). *Given a point a, let Ψ and Γ be the vertical and the horizontal axes passing through it, respectively. We say that a is a* convergence point *for robot r (or that r converges towards a) if, within finite time, r passes any vertical axis to the left of Ψ and any horizontal axis below Γ, and never passes neither Ψ or Γ.*

Note that, by Observation 2, all robots that converge towards a point a are below and to the left of a. The following lemma shows that f is the only converge point.

Lemma 5. *All robots converge towards point f.*

From the previous lemma, and by the termination condition of the algorithm, we can state the following

Corollary 2. *After finite time, all robots terminate their execution, being at distance ε from each other.*

By Corollaries 1 and 2, and by Lemma 4, we can state the following

Theorem 1. *Algorithm 2 correctly solves the* Near-Gathering *problem.*

4.5 On the Knowledge of n

In the solution that we presented, in order for the robots to explicitly terminate, the knowledge of n is necessary. However, this assumption can be dropped by using external visible bits, as recently introduced in [3]. In particular, each robot is equipped with a visible light, whose color can be changed during the *Compute* state. During the *Look*, a robot can retrieve, beside the position, also the value of the light of its fellow robots, which can be stored in a local `Light[]` array (the color of the light of the executing robot is stored in `Light[1]`).

With this extra information, the explicit termination of the robots can be achieved by substituting the termination check in the Near-Gathering protocol with the following check, where ε is an arbitrary small constant (any fraction of V):

If $|Z \setminus Z_\varepsilon| == 0$ **Then**
  `Light`$[\mathtt{r}^*] = 1$;
  If $\forall r \in Z_\varepsilon$, `Light`$[\mathtt{r}] == 1$ **Then Terminate**.
Else `Light`$[\mathtt{r}^*] = 0$.

5 Conclusions

In this paper we presented the first algorithm that solves the Near-Gathering problem for a set of autonomous mobile robots with limited visibility (where the distance function is induced by the infinity norm); the protocol presented here is collision-free: This allows to potentially combine our protocol with solutions designed for the unlimited visibility setting.

We remark that our algorithm also solves the Near-Gathering problem in the robot model that uses the Manhattan distance (i.e., the distance induced by the 1-norm): Each robot merely has to transform each snapshot that it gets during a *Look* state by rotating it clockwise by 45° and scaling it by a factor of $\sqrt{2}$. Then the protocol can be applied as it is, and finally the computed point dp has to be moved again with the inverse transformation: Scaled by $1/\sqrt{2}$ and rotated counterclockwise by 45°.

The Near-Gathering algorithm can also be applied to models that use distances induced by any p-norm, with $p > 1$, including the usual Euclidean distance: Each robot r just "ignores" any point p such that $\|r - p\|_\infty > V$, thus pretending to be in the infinity norm model. Of course, this is guaranteed to terminate correctly only if the initial conditions given in Section 2.1 are met, i.e., if $G(0)$, computed with the infinity norm, is connected.

In particular, when using the Euclidean distance, our protocol and proofs work if $G(t)$ is constructed by connecting pairs of robots that are within Euclidean distance $V/\sqrt{2}$, as opposed to V. Moreover, we are confident that even this constraint on the initial distance graph can be dropped, by a simple adaptation of our protocol to circular visible areas. Due to space limitations, we are unable to discuss the topic further in this paper.

Acknowledgments. We would like to thank Paola Flocchini, Nicola Santoro, and Peter Widmayer, who contributed to the writing of this paper by sharing their ideas.

References

1. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: A distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Transaction on Robotics and Automation 15(5), 818–828 (1999)
2. Cieliebak, M.: Gathering Non-oblivious Mobile Robots. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 577–588. Springer, Heidelberg (2004)
3. Das, S., Flocchini, P., Prencipe, G., Santoro, N., Yamashita, M.: The power of lights: Synchronizing asynchronoys robots using visibile bits. In: The 32nd International Conference on Distributed Computing Systems, ICDCS (to appear, 2012)
4. Défago, X., Souissi, S.: Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. Theoretical Computer Science 396(1-3), 97–112 (2008)
5. Dieudonné, Y., Labbani-Igbida, O., Petit, F.: Circle formation of weak mobile robots. ACM Transactions on Autonomous and Adaptive Systems 3(4) (2008)

6. Dieudonné, Y., Petit, F., Villain, V.: Leader Election Problem versus Pattern Formation Problem. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 267–281. Springer, Heidelberg (2010)
7. Efrima, A., Peleg, D.: Distributed Models and Algorithms for Mobile Robot Systems. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 70–87. Springer, Heidelberg (2007)
8. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of robots with limited visibility. Theoretical Computer Science 337(1-3), 147–168 (2005)
9. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Arbitrary pattern formation by asynchronous oblivious robots. Theoretical Computer Science 407, 412–447 (2008)
10. Katreniak, B.: Convergence with Limited Visibility by Asynchronous Mobile Robots. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 125–137. Springer, Heidelberg (2011)
11. Pagli, L., Prencipe, G., Viglietta, G.: Getting close without touching. Technical Report TR-12-05, Dipartimento di Informatica, Università di Pisa (2012)
12. Peleg, D.: Distributed Coordination Algorithms for Mobile Robot Swarms: New Directions and Challenges. In: Pal, A., Kshemkalyani, A.D., Kumar, R., Gupta, A. (eds.) IWDC 2005. LNCS, vol. 3741, pp. 1–12. Springer, Heidelberg (2005)
13. Prencipe, G.: The effect of synchronicity on the behavior of autonomous mobile robots. Theory of Computing Systems (TOCS) 38(5), 539–558 (2005)
14. Souissi, S., Défago, X., Yamashita, M.: Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. ACM Transactions on Autonomous and Adaptive Systems 4(1), 1–27 (2009)
15. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: formation of geometric patterns. Siam Journal on Computing 28(4), 1347–1363 (1999)
16. Yamashita, M., Suzuki, I.: Characterizing geometric patterns formable by oblivious anonymous mobile robots. Theoretical Computer Science 411(26-28) (2010)

Gathering of Robots on Anonymous Grids without Multiplicity Detection*

Gianlorenzo D'Angelo[1], Gabriele Di Stefano[2], Ralf Klasing[3], and Alfredo Navarra[4]

[1] MASCOTTE project CNRS-INRIA-UNS France
gianlorenzo.d_angelo@inria.fr
[2] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, Italy
gabriele.distefano@univaq.it
[3] CNRS / LaBRI / Universitè Bordeaux 1, France
klasing@labri.fr
[4] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy
alfredo.navarra@unipg.it

Abstract. The paper studies the gathering problem on grid networks. A team of robots placed at different nodes of a grid, have to meet at some node and remain there. Robots operate in Look-Compute-Move cycles; in one cycle, a robot perceives the current configuration in terms of occupied nodes (Look), decides whether to move towards one of its neighbors (Compute), and in the positive case makes the computed move instantaneously (Move). Cycles are performed asynchronously for each robot. The problem has been deeply studied for the case of ring networks. However, the known techniques used on rings cannot be directly extended to grids. Moreover, on rings, another assumption concerning the so-called *multiplicity detection* capability was required in order to accomplish the gathering task. That is, a robot is able to detect during its Look operation whether a node is empty, or occupied by one robot, or occupied by an undefined number of robots greater than one.

In this paper, we provide a full characterization about gatherable configurations for grids. In particular, we show that in this case, the multiplicity detection is not required. Very interestingly, sometimes the problem appears trivial, as it is for the case of grids with both odd sides, while sometimes the involved techniques require new insights with respect to the well-studied ring case. Moreover, our results reveal the importance of a structure like the grid that allows to overcome the multiplicity detection with respect to the ring case.

1 Introduction

In the field of robot based computing systems, one of the most popular problems is certainly the *gathering*. A pool of robots, initially situated at various locations, have to gather at the same place (not determined in advance) and remain there.

* Research supported by the LaBRI under the "Project émergent" program.

G. Even and M.M. Halldórsson (Eds.): SIROCCO 2012, LNCS 7355, pp. 327–338, 2012.

Many variants of the problem have attracted the interest of numerous researchers (see e.g., [1,2] and references therein). In this paper, we consider the case of anonymous grid networks where anonymous, asynchronous and oblivious robots can move according to the so-called Look-Compute-Move cycles [3]. In each cycle, a robot takes a snapshot of the current global configuration (Look), then, based on the perceived configuration, decides either to stay idle or to move to one of its adjacent nodes (Compute), and in the latter case it makes an instantaneous move to this neighbor (Move). Cycles are performed asynchronously for each robot. This means that the time between Look, Compute, and Move operations is finite but unbounded, and is decided by the adversary for each robot. Hence, robots may move based on significantly outdated perceptions. Robots are oblivious, i.e., they do not have any memory of past observations. Thus, the target node (which is either the current position of the robot or one of its neighbors) is decided by the robot during a Compute operation solely on the basis of the location of other robots perceived during the Look operation. Robots are anonymous and execute the same deterministic algorithm. They cannot leave any marks at visited nodes, nor send any messages to other robots. We remark that the Look operation provides the robots with the entire grid configuration concerning occupied nodes. That is, a robot perceives whether a node of the grid is occupied or not, but it cannot distinguish how many robots reside on an occupied node.

Related Work and Our Results. The problem of making mobile entities meet on graphs [3,4,5,6] or open spaces [1,7,8] has been extensively studied in the last decades. When only two robots are involved, the problem is usually referred to as the *rendezvous* [5,9,10,11].Under the Look-Compute-Move model, many problems have been addressed, like the *graph exploration* and the *perpetual graph exploration* [12,13,14,15], while the rendezvous problem has been proven to be unfeasible on rings [3].

Concerning the gathering under the Look-Compute-Move model, much work has been done in the last years for the ring topology. It has been proven that the gathering is unsolvable if the robots are not empowered by the so-called *multiplicity detection* capability [3], either in its *global/strong* or *local/weak* version. In the former type, a robot is able to perceive whether any node of the network is occupied by a single robot or more than one (i.e., a *multiplicity* occurs). In the latter type, a robot is able to perceive the multiplicity only if it is part of it.

Using the global multiplicity detection, different types of configurations have required different approaches. In particular, periodicity and symmetry arguments have been exploited. In a ring, a configuration is called *periodic* if it is invariable under non-trivial (i.e., non-complete) rotations. A configuration is called *symmetric* if the ring has a geometrical *axis of symmetry* that reflects single robots into single robots, multiplicities into multiplicities, and empty nodes into empty nodes. In [3], it is proven that, even with the global multiplicity detection, the gathering is unsolvable for two robots, for periodic configurations and for those symmetric configurations where the axis of symmetry passes through two edges. Then, several algorithms have been proposed for different kinds of initial configurations, in detail: for the case of odd number of robots and that of asymmetric

configurations [3], for symmetric configurations with an even number of robots greater than 18 [16], and for 4 and 6 robots [17,18].

Using the local multiplicity detection in a ring, in [19] it is shown that a configuration is gatherable if $k < \lfloor \frac{n}{2} \rfloor$, while in [20], the case where k is odd and smaller than $n-5$ is studied, where n and k are the number of nodes and robots, respectively. The remaining cases are still open.

In this paper, we fully characterize the gathering on grids. We show that the multiplicity detection capability is not needed. In particular, we show that even if the global multiplicity detection is assumed, a configuration is ungatherable only if it is periodic (i.e., the same view can be obtained by rotating the grid around its geometric center of an angle smaller than 360 degrees) on a grid with at least an even side, or it is symmetric with the axis of symmetry passing through edges. For all the other cases, we provide a gathering algorithm which does not require any multiplicity detection except for configurations on 2×2 grids with three robots where the local multiplicity detection would be helpful.

To our knowledge, the grid topology is the least structured class of graphs that permits to avoid the multiplicity detection assumption. Moreover, it is worth mentioning that in our solutions many robots can move concurrently, instead of just one or two as it was for the ring case.

2 Definitions and Notation

We consider an anonymous and undirected grid of $m \times n$ nodes, with $m \geq n$. Initially, each node is occupied by at most one robot. The total number of robots is denoted by k. During a Look operation, a robot perceives the relative locations on the grid of occupied nodes, regardless of the number of robots on a node.

The current configuration of the system can be described in terms of the view of a robot r which is performing the Look operation at the current moment. We denote a configuration seen by r as an $m \times n$ matrix M on elements in the set $\{0, 1\}$. Value 0 represents an empty node, and 1 represents an occupied node. Note that, if one node is occupied by more than one robot, it is not perceived by the robots, even if they reside on such a node. Since the grid is anonymous and undirected, each robot can perceive the current configuration with respect to different rotations and reflections leading to any view of the grid satisfying the $m \times n$ dimension. In particular, when $m = n$ each of the 4 rotations and 4 reflections provides a feasible view.

Definition 1. *A configuration is* periodic *if it is invariant with respect to rotations of* 90*,* 180*, or* 270 *degrees, where the rotation point coincides with the geometric center of the grid.*

Definition 2. *A configuration is* symmetric *if it is invariant after a reflection with respect to a vertical, horizontal, or diagonal (in case of square grids) axis passing through the geometric center of the grid.*

3 Gathering Algorithms

In this section, we distinguish among different cases concerning the grid structure. In particular, if the grid has both sides odd, the gathering is easily solvable.

If only one side is odd, there are some ungatherable cases. However, the impossibility results do not depend on the assumed multiplicity detection capability. If both sides are even, the gathering strategy relies on the multiplicity detection only if the input grid has size 2×2 and there are three robots, otherwise there is no need of such a capability.

3.1 Odd×odd Grids

This case is trivially solvable, in fact:

Theorem 1. *Configurations on odd×odd grids are always gatherable.*

Proof. In odd×odd grids, a robot can always detect, during its Look operation, the central node of the grid $M[\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil]$, regardless of its possible view. This means that all the robots can move toward the center, concurrently. □

3.2 Odd×even Grids

In this case, the gathering is not always feasible. In fact, similarly to the ring case on periodic or symmetric configurations of type edge-edge [3], we can prove:

Theorem 2. *If a configuration C is periodic, or symmetric with respect to an axis passing through edges (i.e., dividing the grid into two halves from the even side), then C is ungatherable.*

In what follows, we assume that the starting configuration does not belong to the ungatherable configurations specified by the above theorem. Then, we provide an algorithm achieving the gathering without multiplicity detection in all the remaining cases. The idea is to distinguish among the two nodes that are the central nodes of the odd borders of the grid. If m (n, resp.) is odd, then the two mentioned nodes are given by positions $M[1, \lceil \frac{m}{2} \rceil]$ and $M[n, \lceil \frac{m}{2} \rceil]$ ($M[\lceil \frac{n}{2} \rceil, 1]$ and $M[\lceil \frac{n}{2} \rceil, m]$, resp.). The line connecting those two nodes will be denoted as the `NS` line. One of the two extreme nodes on the `NS` line will be the place where the gathering is finalized, eventually. In order to select the gathering node, a robot considers the line passing through the central edges of the even sides of the grid (denoted as the `EW` line) dividing the grid into two halves. The idea is to distinguish a north and a south part among the two halves and the gathering node will be the one in the north half. The north is the half with more nodes occupied by robots, if any. If the number of occupied nodes in the two halves is the same, then some more computations are required (see next paragraph). In both cases, the robots move from the south to the north until all the robots will be in the north part. Note that, during such a stage, if multiplicities are created in the south, then the number of occupied nodes decreases with respect to the north part. If multiplicities are created in the north, it means that a robot has moved from the south to the north part, still preserving the required distinction.

In order to distinguish the north from the south in the case of configurations with the same number of robots among the two halves obtained by the `EW` line, a

robot associates to each configuration C a binary string as follows. Starting from each corner of the grid, and proceeding in the direction parallel to the NS line, a robot records the elements of M row by row, or column by column (according to the direction specified by the NS line). Once it has computed the four strings, it associates to C the lexicographically largest one. For instance, starting from corner $M[1,1]$, and assuming m odd, the corresponding binary string would be composed by the sequence $M[1,1]$, $M[2,1]$, ..., $M[n,1]$, $M[1,2]$, ..., $M[n,2]$, $M[1,m]$, ..., $M[n,m]$.

Lemma 1. *Let C be a gatherable configuration, then, among the four possible strings coming from a robot view of the input grid, at most two strings can be the lexicographically largest ones. If there are two largest strings, then they represent the views of C starting from two symmetric corners with respect to the NS line.*

Proof. If the equal strings correspond to the view of C starting from two symmetric corners with respect to the EW line, then C would be symmetric with respect to the EW line. In fact, from Definition 2, this would correspond to a reflection of the grid with respect to the EW line. But, from Theorem 2, it would imply that C is ungatherable. If the equal strings correspond to the view of C starting from two corners residing on one of the two diagonals of the grid, then C would be periodic. In fact, from Definition 1, this would correspond to a rotation of the grid of 180 degrees, again despite C being gatherable. Then, no more than two strings can be equal as otherwise one of the above situations would occur. It follows that either the four strings are all different among themselves, or there are two pairs of equal strings, one of which corresponds to the lexicographically largest ones. Moreover, both correspond to the view of C starting from two symmetric corners with respect to the NS line. □

From the above lemma, we define the *gathering node* as the one residing on the same odd side where the corner(s) providing the lexicographically largest string resides. Moreover, the gathering node will determine also the directions along the NS line: We say that the gathering node resides on the north pole.

Theorem 3. *Configurations on odd×even grids that are aperiodic and do not admit an axis of symmetry passing through edges are always gatherable.*

Proof. Once the gathering node has been unambiguously identified by a robot during its Compute operation, if the robot resides on the half grid where the south pole is, with respect to the EW line, then it moves towards the north. Note that, each time a robot in the southern half of the grid performs such a movement, the gathering node cannot change. In fact, two cases can occur: 1) the number of occupied nodes decreases in the southern part of the grid, either because a robot moves to the northern part or because a multiplicity is created; 2) the string associated to the corners in the south are decreasing due to the robots' movements, and hence the corresponding strings defining the current configuration starting from the northern corners are increasing. This clearly leaves unchanged the direction on the NS line. Note that the corner to which the lexicographically largest string was associated might change during the described

process but the only option is the other corner on the same odd side of the original one, hence preserving the direction on the NS line. By keeping on moving in the described way, all the robots will reach the northern part, eventually. The case in which a subset of robots from a multiplicity move, increasing the number of occupied nodes, does not require any special treatment. In fact, since the initial configuration does not contain multiplicities, either the minimality of the number of robots in one half of the grid is preserved, or case 2) still ensures that the lexicographically largest string is associated to a corner in the north.

Once all the robots belong to one half of the grid, then they are allowed to move, during their Move operation, towards the gathering node. In fact, such a node is now well-defined and cannot change as the robots are not allowed to move to the other half of the grid. □

3.3 Even×even Grids

In this section, we study the case of grids whose sides are both even. Also in this case, by Theorem 2, there are some configurations which are ungatherable, namely the periodic configurations and those configurations having a vertical or a horizontal axis of symmetry. We show that all the other cases are gatherable without any multiplicity detection, but for the case of 2×2 grids.

Theorem 4. *Let us consider a 2×2 grid with more than one node occupied. If the multiplicity detection is not allowed, then any configuration is ungatherable. If the local multiplicity detection is allowed, a configuration is gatherable if and only if it has three nodes occupied.*

Proof. Clearly, a 2×2 grid equals a ring of dimension four. Hence, any ungatherable configuration on a ring of four nodes is ungatherable on a 2×2 grid. In particular, configurations with two or four nodes occupied are ungatherable even with the global multiplicity detection and configurations with three nodes occupied are ungatherable if the multiplicity detection is not allowed [3].

Finally, we show that a configuration is gatherable with the local multiplicity detection if three nodes are occupied. Consider the following algorithm:

1. move the robot in between the other two occupied nodes arbitrarily;
2. move the robot not in the multiplicity towards the other occupied node. □

Hence, the remaining gatherable configurations are the aperiodic, asymmetric, and those with only one axis of symmetry passing through the diagonal of a square grid of dimensions larger than 2×2. We refer to all such configurations as the set EG (Even-Gatherable). In Theorem 5, we will show that all the configurations in EG are indeed gatherable without any multiplicity detection.

In the following, we first assume that at least one node on the border of the grid is occupied. Then, in the proof of Theorem 5, we will show how to extend the given strategies to the general case. First, we give some definitions about the "reading" of grid configurations needed for the subsequent proofs.

Let us consider the eight sequences of distances (number of empty nodes) between occupied nodes obtained by traversing the grid starting from the four

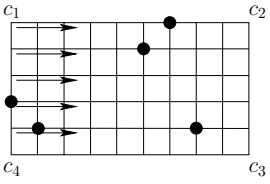


Fig. 1. Case of a 10×6 grid. The arrows indicate the horizontal direction of the reading from corner c_1, it gives $(6, 8, 14, 10, 5, 12)$. The other seven sequences read by the robots are: $(3, 6, 20, 4, 9, 13)$ from c_1 vertically, $(3, 10, 24, 2, 5, 11)$ and $(16, 1, 6, 26, 4, 2)$ from c_2 horizontally and vertically, resp., $(12, 5, 10, 14, 8, 6)$ and $(13, 9, 4, 20, 6, 3)$ from c_3, $(11, 5, 2, 24, 10, 3)$ and $(2, 4, 26, 6, 1, 16)$ from c_4. The *minimal* sequence is $(2, 4, 26, 6, 1, 16)$ and $c = c_4$.

corners and proceeding towards the two possible directions (see, e.g. Fig. 1). Note that the two sequences associated to a corner occupied by some robot starts with 0. We associate for each corner the lexicographically smallest sequence between the two readings from such corner. Note that, in square grids such two sequences are always different, but for the two corners through which passes the possible axis of symmetry. In rectangular grids, these two sequences can be equal but we can distinguish one of them by assuming that if two sequences are equal, the one read in the direction of the smallest side is smaller than the other.

We define the *minimal* sequence as follows. If the configuration is symmetric, it is the smallest sequence between the two sequences associated to the two corners through which passes the axis of symmetry, otherwise it is the smallest among the four sequences associated to the four corners. Note that, under the assumption that the configuration does not fall into the hypothesis of Theorem 2, in any case there exists a minimal sequence which identifies a single corner, unambiguously. We denote the minimal sequence as $C = (q_0, q_1, \ldots, q_j)$ and by c the corner which it is associated to.

An important property of our gathering strategy that we are going to present is: *In all the movements used in the following results we do not allow a robot to move into a corner different from c.*

Lemma 2. *For any* `EG` *configuration with no corners occupied and at least one robot on the border there exists a strategy that leads to a configuration with exactly one corner occupied.*

Proof. If there are no corners occupied, the idea is to reduce q_0 by moving towards c the robot (or the two robots, when the configuration is symmetric) on the border which is (are) closest to c. Note that, as we are assuming that there is at least a node on the border and that C is the minimal sequence, the robot (robots) involved in the movement towards c is (are) on the border.

In the case of symmetric configurations, we aim to move towards c the two symmetric robots on the border which are the closest ones to c. Let us denote these two robots by r and r'. First of all, we prove that if only one robot moves

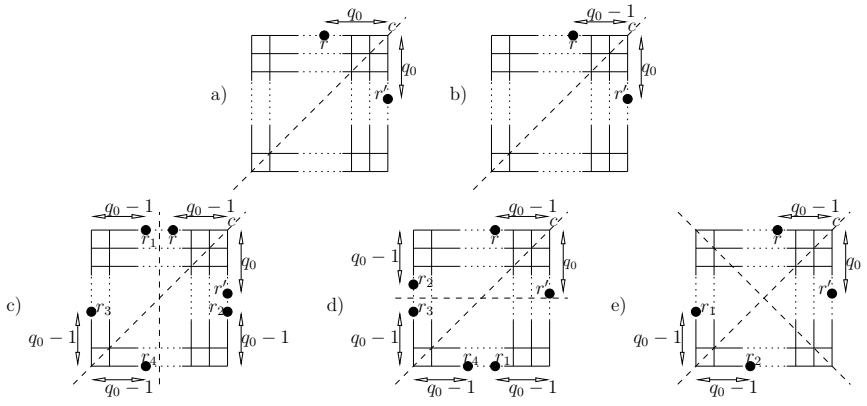


Fig. 2. Symmetric square grid with no corners occupied. Dashed lines represent axes of symmetry. a) original configuration; b) configuration once only robot r has moved; c) configuration with a possible vertical axis; d) configuration with a possible horizontal axis; e) configuration with a possible diagonal axis different from the original one.

(let us assume r), no symmetric configuration can be created and, moreover, there exists only one robot (r') at one allowed move from a symmetric configuration. We prove this by showing that contradicting such statement would imply that, in the initial configuration, the sequence associated to the corner on the axis different from c starts with $q_0 - 1$ (see Fig. 2), which is a contradiction with respect to the minimality of C. Moving robot r may create three potential axes of symmetry which are, with respect to the drawing of Fig. 2: a vertical, horizontal, or diagonal axis different from the original one. Let us assume that there exists a vertical axis of symmetry (see Fig. 2c). In this case the presence of the vertical axis of symmetry and of robot r at distance $q_0 - 1$ from c, implies the presence of robot r_1 of Fig. 2c which is specular to r with respect to the vertical axis of symmetry. Note that, in this case, $q_0 - 1 < \frac{n}{2}$ as otherwise r_1 would be closer to c than r, a contradiction. Therefore, r_1 is at distance $q_0 - 1$ from the upper left corner of Fig. 2c. Now, since this robot did not move, the original diagonal axis of symmetry implies the presence of a robot r_2 at distance $q_0 - 1$ from the lower right corner of Fig. 2c. Again, the two axes imply the presence of robots r_3 and r_4, both at distance $q_0 - 1$ from the corner opposite to c, a contradiction to the minimality of C. The case of horizontal axis of symmetry is similar and it is shown in Fig. 2d. Let us now assume that there exists a diagonal axis of symmetry different from the original one (see Fig. 2e). Such axis implies the presence of robot r_1 at distance $q_0 - 1$ from the corner opposite to c. Moreover, the original axis implies the presence of robot r_2 at distance $q_0 - 1$ from the corner opposite to c. Again, this is a contradiction to the minimality of C. Similar arguments can be used to show that there cannot exist other robots besides r' at one allowed move from a symmetric configuration. Hence, in the

case of symmetric configurations, or asymmetric configurations at one allowed move from symmetry, the strategy leads to the occupation of c, possibly with a pending move towards c.

In the case of asymmetric configurations, first of all the robots check whether the configuration is at one step from a symmetric configuration belonging to EG that can be obtained by the move performed by a robot (potentially corresponding to r') on the border towards the corner c that would be obtained when the axis of symmetry occurs. We recall that, in the case of symmetric configurations, c is defined as the corner whose associated sequence is the smallest one among the two corners lying on the axis of symmetry and not among all the corners. If such a symmetry cannot be established, then c corresponds to the minimal sequence and the algorithm proceeds by reducing q_0 until it becomes 0, that is, c is occupied. In fact, if the initial configuration is asymmetric and $q_0 > 1$, then after reducing q_0, the obtained configuration is again asymmetric as the minimal sequence of the new configuration starts with $q_0 - 1$ while any other sequence starts with at least q_0. When the initial configuration is asymmetric and $q_0 = 1$, after the move, the configuration might become symmetric but with one corner occupied. In conclusion, in any case the obtained configuration has the corner c occupied, and possibly one pending move towards c. □

Before showing the case of configurations in EG with two corners occupied, we need to exclude all the configurations with exactly three occupied nodes, two of which are two corners that share a coordinate, and the other one is at one node apart from another corner. See the configuration in the middle of Fig. 3. We denote such configurations as the set 3EG2.

Lemma 3. *For any configuration in EG\3EG2 with two corners occupied there exists a strategy that leads to a configuration with either exactly one corner occupied or exactly three corners occupied.*

Proof. If two corners are occupied, and the configuration is symmetric with the axis passing through the occupied corners, then one of them corresponds to c. We move the robot in the occupied corner which does not corresponds to c towards the other one, and we end up with the case of only one corner occupied.

The case where two corners are occupied, and the configuration is asymmetric or symmetric with the axis not passing through the occupied corners requires some more effort. In this case, it is risky to move the robots from the occupied corners, since if the adversary forces to move only one of them, we could not be able to recognize the possible move of the other robot which is still pending. Therefore, let d be the corner not occupied by a robot from which we read the minimal sequence D. We move towards d the first robot not in a corner that reduces D, and possibly, the symmetric one.[1] By repeating this strategy, d will be occupied by at least one robot, eventually. Arguments similar to those used in the proof of Lemma 2 can be used to show that no symmetries different from the original one can be created. It follows that for each step in which this strategy is

[1] Here we do not need to preserve the symmetry as it was necessary in Lemma 2.

applied, d remains the same. The final configuration has three corners occupied and possibly one pending move towards d of a robot not in a corner. □

Note that the configuration obtained after the strategy given in Lemma 3 is always in EG with possibly one pending move.

Theorem 5. *Aperiodic configurations on even×even grids larger than 2×2, that do not admit an axis of symmetry passing through edges, are gatherable.*

Proof. First, we can restrict the set of possible grids as follows. Let us consider the minimal even×even sub-grid which is centered in the geometrical middle of the original grid and includes all the occupied nodes of it. Such minimal *wrapping* grid is still of type even×even and preserves the possible symmetry of the original one. Moreover, it always has at least an occupied node on the border. Our algorithm only uses such sub-grids and it never changes the size of it, i.e. it neither enlarges it by moving robots outside of it, nor it reduces it by moving the robots on the border inside. Therefore, in what follows, we can assume without loss of generality that a grid always has at least an occupied node in the border. However, the case in which the resulting wrapping grid is a 2×2 grid will be considered separately.

If no corners are occupied, we can apply Lemma 2 that leads to a configuration with one corner occupied with possibly one pending move towards c. However, such a move would have been performed also in the strategy used for the case of one corner occupied which is given in the following.

If only one corner is occupied, then it corresponds to c. In this case, all the robots move towards c by reducing the Manhattan distance to c and then achieving the gathering. We remind that the allowed movements are performed always without occupying any other corner than c.

In case two corners are occupied but the configuration is not in 3EG2, we apply the strategy of Lemma 3 and reach a configuration with one or three corners occupied. Also in this case there could be a pending move and, again, such a move would have been performed also in the strategy for one corner or three corners occupied.

If three corners are occupied, we move all the robots, but those in the corners, towards the corner that does not share any coordinate with the empty corner. This process finishes with a symmetric configuration with exactly three corners occupied. In this configuration, c is the corner on the axis of symmetry, and the other two robots move one step towards c either concurrently or alternately, until creating a configuration with only one corner occupied as shown in Fig. 3. Note that these final steps also solve the gathering for the configurations in 3EG2.

If four corners are occupied, we move the robot which occupies the corner farthest from c in an arbitrary direction, generating a configuration where only three corners are occupied.

It remains the case where the minimal wrapping even×even sub-grid which includes all the occupied nodes of the original grid has dimension 2×2. As shown in Theorem 4, the configuration is not gatherable on this sub-grid without multiplicity detection. However, in the case of exactly three nodes occupied, we can

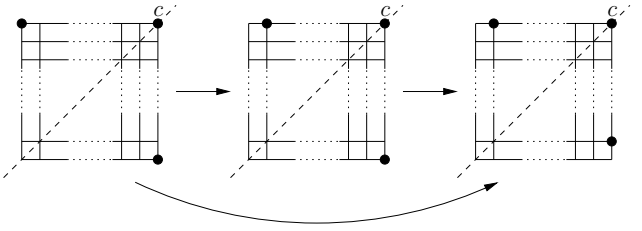


Fig. 3. Strategy from a symmetric square grid where exactly three corners are occupied and all the other nodes are empty to a configuration with only one occupied corner, possibly passing through configurations in 3EG2

exploit the larger dimensions of the original grid in order to avoid the multiplicity detection. The cases of two or four nodes occupied clearly remain not gatherable. The strategy is then to move the robot on the corner of the 2×2 grid which is in between the other two occupied corners towards the external row or column, arbitrarily. In doing so, we obtain the case where the minimal wrapping grid has dimension 4×4 and no corners are occupied. □

4 Conclusion

We fully characterized the gathering in the Look-Compute-Move model on grids. We have shown that a configuration is ungatherable if and only if it is periodic on a grid with at least an even side, or it is symmetric with the axis passing through edges, or it is a 2×2 grid. For all the other cases we provided a gathering algorithm which does not require any multiplicity detection. It would be of interest to investigate whether the grid topology is the least structured class of graphs that permits to avoid the multiplicity detection assumption.

References

1. 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: Proc. of the 23rd ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 139–148 (2011)
2. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theor. Comput. Sci. 337, 147–168 (2005)
3. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theor. Comput. Sci. 390, 27–39 (2008)
4. Bampas, E., Czyzowicz, J., Gąsieniec, L., Ilcinkas, D., Labourel, A.: Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 297–311. Springer, Heidelberg (2010)
5. Dessmark, A., Fraigniaud, P., Kowalski, D., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica 46, 69–96 (2006)

6. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Randomized Gathering of Mobile Robots with Local-Multiplicity Detection. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 384–398. Springer, Heidelberg (2009)
7. Cord-Landwehr, A., Degener, B., Fischer, M., Hüllmann, M., Kempkes, B., Klaas, A., Kling, P., Kurras, S., Märtens, M., Meyer auf der Heide, F., Raupach, C., Swierkot, K., Warner, D., Weddemann, C., Wonisch, D.: A New Approach for Analyzing Convergence Algorithms for Mobile Robots. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 650–661. Springer, Heidelberg (2011)
8. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theor. Comput. Sci. 384, 222–231 (2007)
9. Alpern, S.: The rendezvous search problem. SIAM J. Control Optim. 33, 673–683 (1995)
10. Chalopin, J., Das, S.: Rendezvous of Mobile Agents without Agreement on Local Orientation. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 515–526. Springer, Heidelberg (2010)
11. Czyzowicz, J., Labourel, A., Pelc, A.: How to meet asynchronously (almost) everywhere. In: Proc. of the 21st Annual ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 22–30 (2010)
12. Blin, L., Milani, A., Potop-Butucaru, M., Tixeuil, S.: Exclusive Perpetual Ring Exploration without Chirality. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 312–327. Springer, Heidelberg (2010)
13. Devismes, S., Petit, F., Tixeuil, S.: Optimal Probabilistic Ring Exploration by Semi-synchronous Oblivious Robots. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 195–208. Springer, Heidelberg (2010)
14. Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Computing without communicating: Ring exploration by asynchronous oblivious robots. Algorithmica (to appear)
15. Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Remembering without memory: Tree exploration by asynchronous oblivious robots. Theor. Comput. Sci. 411(14-15), 1583–1598 (2010)
16. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring. Theor. Comput. Sci. 411, 3235–3246 (2010)
17. D'Angelo, G., Di Stefano, G., Navarra, A.: Gathering of Six Robots on Anonymous Symmetric Rings. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 174–185. Springer, Heidelberg (2011)
18. Koren, M.: Gathering small number of mobile asynchronous robots on ring. Zeszyty Naukowe Wydzialu ETI Politechniki Gdanskiej. Technologie Informacyjne 18, 325–331 (2010)
19. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Mobile Robots Gathering Algorithm with Local Weak Multiplicity in Rings. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO 2010. LNCS, vol. 6058, pp. 101–113. Springer, Heidelberg (2010)
20. Kamei, S., Lamani, A., Ooshita, F., Tixeuil, S.: Asynchronous Mobile Robot Gathering from Symmetric Configurations without Global Multiplicity Detection. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 150–161. Springer, Heidelberg (2011)

Author Index