Shay Kutten
Janez Žerovnik (Eds.)

LNCS 5869

Structural Information and Communication Complexity

16th International Colloquium, SIROCCO 2009
Piran, Slovenia, May 2009
Revised Selected Papers

Springer

Lecture Notes in Computer Science 5869

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

Shay Kutten Janez Žerovnik (Eds.)

Structural Information and Communication Complexity

16th International Colloquium, SIROCCO 2009
Piran, Slovenia, May 25-27, 2009
Revised Selected Papers

Springer

Volume Editors

Shay Kutten
Technion – Israel Institute of Technology
Dept. of Industrial Engineering and Management
Technion City, Haifa 32000, Israel

Janez Žerovnik
University of Ljubljana
Faculty of Mechanical Engineering
Aškerčeva 6, 1000 Ljubljana, Slovenia
E-mail: janez.zerovnik@imfm.uni-lj.si

Library of Congress Control Number: 2010920246

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

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

ISSN 0302-9743
ISBN-10 3-642-11475-X Springer Berlin Heidelberg New York
ISBN-13 978-3-642-11475-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12837144 06/3180 5 4 3 2 1 0

Preface

These are the proceedings of SIROCCO 2009: the 16$^{\text{th}}$ annual Colloquium on Structure, Information, Communication, and Complexity. SIROCCO is devoted to the study of the interplay and trade-offs between the efficiency of decentralized algorithms and systems and the availability of information.

Over the years, the colloquium has become a widely recognized forum, bringing together researchers interested in the fundamental principles underlying the interplay between local knowledge and global complexity. It has a tradition of interesting and productive scientific meetings in a relaxed and pleasant atmosphere, attracting leading researchers in a variety of fields which exhibit such interplay.

This means that SIROCCO addresses topics in areas such as distributed computing, parallel computing, game theory, social networks, networking, mobile computing, peer to peer systems, communication complexity, combinatorial optimization, etc. Some of the topics in these areas are compact data structures, information dissemination, informative labeling schemes, distributed scheduling, wireless networks and scheduling of transmissions, routing, broadcasting, localization, and others.

SIROCCO 2009 was held in Piran, Slovenia, on the Adriatic. There were 53 contributions submitted to SIROCCO 2009. The submissions underwent a thorough refereeing process, where each submission was reviewed by four members of the Program Committee. After in-depth discussions, the Program Committee selected 23 high-quality contributions for presentation at the colloquium and publication in this volume. Separately, four posters were also presented (but they are not included in these proceedings). We thank the authors of all the submitted papers, the Program Committee members, and the external reviewers. Without their dedication, we could not have prepared a program of such quality.

There were two invited speakers: Israel Cidon (the Technion) and Leszek A. Gasieniec (University of Liverpool).

This year, the SIROCCO Prize for Innovation in Distributed Computing was awarded for the first time. The prize was given to Nicola Santoro for his overall contribution on the analysis of the labeled graph properties, which has been shown to have a significant impact on computability and complexity in systems of communicating entities. These contributions include the notions of "implicit routing," "sense of direction," and "topological awareness." They were illustrated by several papers, including his 1994 SIROCCO paper.

We express our gratitude to the SIROCCO Steering Committee, and in particular to Pierre Fraigniaud for his enthusiasm and his invaluable help throughout the preparation of this event.

We are in a great debt to Igor Pesek who helped in many ways, including handling the website and EasyChair. Petra Šparl was also very instrumental in making SIROCCO 2009 a success.

We acknowledge the use of the EasyChair system for handling the submission of papers, managing the refereeing process, and generating these proceedings.

August 2009

Shay Kutten
Janez Žerovnik

Organization

Program Committee

Jofroy Beauquier	University of Paris-Sud
Shantanu Das	ETH
Pascal Felber	University of Neuchâtel
Chryssis Georgiou	University of Cyprus
Seth Gilbert	EPFL
David Ilcinkas	CNRS and University of Bordeaux
Ralf Klasing	CNRS and University of Bordeaux
Spyros Kontogiannis	University of Ioannina
Miroslaw Korzeniowski	Wroclaw University of of Technology
Dariusz Kowalski	University of Liverpool
Shay Kutten (Chair)	Technion
Emmanuelle Lebhar	CNRS and University of Chile
Zvi Lotker	Ben Gurion University
Toshimitsu Masuzawa	Osaka University
Jaroslav Opatrny	Concordia University
Giuseppe Persiano	University of Salerno
Linda Pagli	Pisa University
Tomasz Radzik	King's College London
Christian Schindelhauer	University of Freiburg
Stefan Schmid	Technische Universität München
Srikanta Tirthapura	Iowa State University
Masafumi Yamashita	Kyushu University
Janez Žerovnik	IMFM and University of Ljubljana

Steering Committee

Felber Pascal	University of Neuchâtel
Flocchini Paola	University of Ottawa
Fraigniaud Pierre (Chair)	CNRS and University of Paris 7
Gasieniec Leszek	University of Liverpool
Kirousis Lefteris	University of Patras
Kranakis Evangelos	Carleton University
Kralovic Rastislav	Comenius University
Krizanc Danny	Wesleyan University
Mans Bernard	Macquarie University
Peleg David	Weizmann Institute
Prencipe Giuseppe	Pisa University
Santoro Nicola	Carleton University

Shvartsman Alex	MIT and University of Connecticut
Spirakis Pavlos	CTI
Zaks Shmuel	Technion

Organizing Committee

Janez Žerovnik (Chair)	IMFM and University of Ljubljana
Petra Šparl	IMFM and University of Maribor
Igor Pesek	University of Maribor

Referees

Auletta Vincenzo
Avin Chen
Bienkowski Marcin
Blaskiewicz Przemyslaw
Böckenhauer Hans-Joachim
Bonuccelli Maurizio
Caragiannis Ioannis
Chalopin Jérémie
Clement Julien
Cohen Reuven
De Marco Gianluca
Disser Yann
Dvir Amit
Elsässer Robert
Esperet Louis
Eyraud-Dubois Lionel
Gasieniec Leszek
Gavoille Cyril
Gebala Maciej
Gramoli Vincent
Grossi Roberto
Hanusse Nicolas
Jez Artur
Jez Lukasz
Kakugawa Hirotsugu
Kellett Matthew
Kik Marcin
Klonowski Marek
Kosowski Adrian
Kralovic Richard
Labourel Arnaud
Lahiri Bibudh
Markou Euripides
Martin Russell
Mihalak Matus
Mordechai Shalom
Moscardelli Luca
Musial Peter
Nicolaou Nicolas
Nisse Nicolas
Ooshita Fukuhito
Penna Paolo
Peterin Iztok
Pucci Geppino
Rawitz Dror
Riviere Etienne
Rokicki Mariusz
Rosaz Laurent
Rozoy Brigitte
Sramek Rastislav
Struminski Tomasz
Thraves Christopher
Ventre Carmine
Xu Bojian
Yamauchi Yukiko
Zawada Marcin
Zylinski Pawel

Table of Contents

Invited Talks

Regular Papers

Zooming in on Network-on-Chip Architectures

Israel Cidon

Electrical Engineering Faculty, Technion, Israel Institute of Technology

Abstract. The aim of this talk is to expose the theoretical distributed system community to the concept of Network-on-Chip (NoC), an emerging research field within the VLSI realm, in which networking principles play a significant role, and new network architectures are being explored in a new setup as well as new cost and performance models. Researchers should find new challenges in exploring solutions to familiar problems such as network design, routing, and quality-of-service, in unfamiliar settings under new constraints. The unique characteristics of silicon chips require new solutions to these classical problems, and define a new set of NoC specific problems, such as automatic network design process, power and area optimization and specialized system functionalities.

We present a new classification of chip architectures into three categories with different requirements from their NoCs. In order to stimulate some research directions, we highlight several research problems arising in these categories such as routing, quality-of-service, flow and congestion control, and resource allocation (e.g., capacity assignment, sharing hot-spots). We provide initial solution directions to example problems.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, p. 1, 2010.

On Efficient Gossiping in Radio Networks

Leszek Gąsieniec*

Abstract. A communication network is very often modelled as a graph of connections in which the nodes exchange information (messages) via (un)directed links. An associated communication protocol determines the way the messages are exchanged. Among the most popular network models are: (1) the *message passing model* in which a node in one round can inform all its neighbours; (2) the *telephone model* also known as the *matching model* where in each round edges along which the exchange of messages is performed form a matching in the graph of connections. More recently, due to arrival of wireless technology (3) the *radio network model* attracted more attention in algorithms community. In this model, a message transmitted by a node is destined for all neighbours of this node. It is assumed, however, that due to interference a node can successfully receive a message if and only if exactly one of its neighbours transmits during this round.

The two most fundamental problems in relation to information dissemination are: *broadcasting* (one-to-all communication) and *gossiping* (total information exchange). In broadcasting, the goal is to distribute a piece of information (*broadcast message*) from a distinguished source node to all other nodes in the network. In gossiping, however, each node in the network is expected to distribute its own message to every other node in the network. A lot of attention has been given to the broadcasting problem that resulted in a large volume of efficient algorithmic solutions in the models described above. However, much less is known about gossiping. The latter problem is more complex algorithmically (in principle it is a simultaneous multiple-source broadcasting) thus it concerns more advanced communication strategies. Further study on efficient gossiping methods gained recently an extra motivation through an increasing interest in, e.g., *information aggregation* methods that propel fundamental applications in sensor networks. Also when the use of randomisation is permitted gossiping provides an interesting context for a distributed version of the *coupon collector problem*.

This paper is a short survey on the most important developments in efficient radio gossiping. We discuss deterministic as well as randomized methods of communication in the context of a variety of models taking into account knowledge in relation to the network size and topology, orientation of connections and the upper bound on the size of messages. Using this opportunity we also shed more light on several combinatorial structures and algorithmic solutions that emerged during studies on efficient radio broadcasting and gossiping.

* Department of Computer Science, University of Liverpool, Ashton Street, Liverpool, L69 3BX, UK. E-mail: {L.A.Gasieniec}@liverpool.ac.uk. This research was partially funded by the Royal Society International Joint Project, IJP - 2007/R1.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 2–14, 2010.

1 Introduction

A *radio network* consists of a number of communication devices, each of which can act at a given time step either as a *transmitter* or as a *receiver*. We very often model a radio network as a (un)directed *graph* $G(V, E)$ *of connections* with nodes in the set V representing communication devices. The nodes in V are labelled by distinct integers drawn from the range $[1, .., N = O(n^c)]$, for a constant $c \geq 1$. If $c = 1$ we say that the labels are *small*, otherwise the labels are referred to as *large*. Two nodes in V are connected by an edge (or an arc) in E if they can communicate directly. Such nodes are referred to as *neighbours*. Nodes located in G at larger distances must communicate via intermediate nodes. The number of nodes $|V| = n$ is considered to be the size of the radio network since presence of edges in E is only virtual. Other important parameters of the graph of connections include the maximum degree Δ and the *diameter* D.

The network nodes communicate with their neighbours using the *radio network protocol* introduced in [5], where full synchronisation of network nodes is assumed. More precisely, the nodes have individual clocks that tick at the same rate, measuring time steps, sometimes referred to as *rounds*. The *running time* of a communication procedure refers to the number of time steps required to accomplish a specific communication task such as broadcasting and gossiping.

A node acting as a transmitter in a given time step sends a message which is delivered to all of its neighbours on the conclusion of the same time step. An important distinction at the receiving end is between a message being *delivered* and being *heard* (decoded, properly recognised), i.e., received successfully by a destination node. It is assumed that a node v acting as a receiver in a given step *hears* a message if and only if a message from exactly one of its neighbours is delivered at this time step. Otherwise, if messages from at least two neighbours are delivered to v simultaneously, none of the messages is heard by v in this step. In this case we say that a *collision* (caused by interference) occurred at v. It is assumed that nodes cannot distinguish collisions from the background noise.

In our presentation of efficient gossiping methods we will first assume that the network topology is *unknown*, i.e., that initially the nodes are not aware of the topology of connections. Such networks are referred to as *ad-hoc radio networks*. In this model the decision made by a node on whether and what to transmit or alternatively whether to receive in a given round, is based solely on the label of the node, the messages it heard so far, and the number of the current round. In ad-hoc radio networks due to lower bounds $\Omega(n \log D)$ for deterministic broadcasting [14] and $\Omega(n \log n)$ for gossiping enforced by the minimum size of *selective families* [9], the main objective is to look for communication procedures with the time complexity almost linear in n .

In the second part of the paper we focus our attention on the model in which a complete topology of connections is known in advance. In this model the emphasis is on the design of communication schedules with the time complexity proportional to the diameter D and the maximum degree Δ of the network.

2 Ad-Hoc Radio Networks

In this section we discuss a number of communication methods that proved to be useful in time efficient radio gossiping in ad-hoc networks. We start with a short review of the most important developments in radio broadcasting.

2.1 Broadcasting

One of the first non-trivial results in this model is $O(D \log n + \log^2 n)$-time randomised broadcasting procedure due Bar-Yehuda *et al.* [2]. Corresponding components of the lower bound of size $\Omega(D \log(n/D))$ and $\Omega(\log^2 n)$ can be found in [40] and in [1] respectively. A tight upper bound $O(D \log(n/D) + \log^2 n)$ was obtained independently by Kowalski and Pelc in [38] and Czumaj and Rytter in [15].

In deterministic broadcasting the starting point is a folklore type quadratic time Round-Robin procedure in which each node transmits on its own periodically. The first non-trivial result on deterministic broadcasting can be found in [8] where Chlebus *et al.* show how to broadcast messages in time $O(n^{11/6})$. In order to cope with simultaneous transmissions their algorithm utilises *selective families* constructed on the basis of the *deterministic sample* introduced by Vishkin in the context of efficient parallel string matching [46]. This upper bound was later improved to $O(n^{5/3} \log^3 n)$ by De Marco and Pelc in [19] and to $O(n^{3/2}\sqrt{\log n})$ by Peleg in [43] who also pointed out the difference between small and large labels of the network nodes. Further, Chlebus *et al.* in [7] developed several broadcasting algorithms, including the one with the time complexity $O(n^{3/2})$ based on arithmetic over a finite field.

The breakthrough in deterministic radio broadcasting came in [11] where Chrobak *et al.* proved existence of small selective families, referred to as *k-selectors*, with a linear selectivity. The definition of k-selectors differs from the definition of $(k, m, N)-$selectors introduced in [17] in the context of *combinatorial group testing problem*, their meaning, however, is comparable. In fact, one can prove that *k-cover-free families* [23], *disjunctive codes* [20], *superimposed codes* [37], and *strongly selective families* [14] correspond to the notion of a $(k+1, k+1, n)$-selector. In particular, k-selectors from [11] coincide with the definition of $(2k, 3k/2+1, n)$-selectors.

The $(k, m, N)-$selectors can be pictured as rectangular matrices with N columns standing for labels from the range $[1, .., N = O(n^c)]$, and rows representing characteristic vectors of suitably chosen subsets of $\{1, .., N = O(n^c)\}$.

Definition 1 ([17]). *For any integer $1 \leq m \leq k < N$, a Boolean matrix M with t rows and n columns is a (k, m, N)-selector if any submatrix of M obtained by choosing k out of n arbitrary columns of M contains at least m distinct rows of the identity matrix I_k. The integer t is the size of the (k, m, N)-selector.*

Assuming that there are k nodes that compete to inform one of their neighbours the use of a selector provides an opportunity to m of them to transmit on their

own. In view of bounds provided in [17] and [9] we know that (k, m, N)-selectors are of size $\Theta(k^2 \log(n/k)/(k-m+1))$, for most of reasonable values of k, m and N. More recent work on the size of selectors include [22,4].

The arrival of selectors of almost linear size in k resulted in development of almost linear time radio broadcasting algorithms. The sequence of transmissions performed by each node depends on the content of a logarithmic number of, e.g., $(2k, 3k/2+1, N)$-selectors of geometrically increasing sizes where the content of selectors is evenly distributed in time. The broadcast process is split into virtual stages, where at each stage the nodes form three groups: (1) containing all informed nodes but with all their neighbours already informed (this group is initially empty); (2) containing all nodes that are informed but still have some uninformed neighbours; (3) containing uninformed nodes. Assume that at the beginning of some stage the size of the group (2) is l and it satisfies a condition $k \leq l \leq 3k/2$ for one of the used selectors. Note that due to the property of $(2k, 3k/2+1, N)$-selectors either a fraction of l nodes have a chance to transmit on their own, i.e., they will be transferred to group (1) or the size of the group (2) grows above $2k$ when the selector becomes not selective enough. However this time there must be a linear in l transfer of nodes from group (3) to group (2). Thus in either of these cases in time $O(l \log^2 n)$, where $O(l \log n)$ comes from the size of the respective $(2k, 3k/2+1, N)$-selector and a multiplicative factor $\log n$ results from simultaneous use of a logarithmic number selectors, $\Omega(l)$ transfers are obtained. Thus to accomplish broadcasting one needs at most $2n$ transfers between the groups the total time complexity of the algorithm proposed in [11] is $O(n \log^2 n)$. Further improvements on the time complexity of radio broadcasting can be found in [38] and [15] with times $O(n \log n \log D)$ and $O(n \log^2 D)$ respectively, and in very recent work of De Marco [18] with the time $O(n \log n \log \log n)$.

2.2 Gossiping

Note that the utilisation of the Round-Robin procedure provides also a solution to the gossiping problem. Unfortunately, it works in time $O(n^2)$ and only for small labels with $N = O(n)$. The first non trivial attempt has been made by Chrobak *et al.* in [11]. Their solution is still based on the Round-Robin principle however the time complexity of gossiping is reduced to $O(n^{3/2} \log^2 n)$.

Algorithm GOSSIP;
perform $\sqrt{n} \log^2 n$ rounds of ROUNDROBIN;
while $\max_v |K(v)| > 0$ **do**
    FIND a node $v_{\max}$, s.t., $|K(v_{\max})| = \max_v |K(v)|$;.
    BROADCAST from $v_{\max}$ message $K(v_{\max})$;
    for each node v
        $K(v) \leftarrow K(v) - K(v_{\max})$;.

The algorithm runs in two stages. During the first stage each message is distributed to at least $\sqrt{n} \log^2 n$ nodes in the network with the help of the Round-Robin procedure. In the second phase, during each iteration we choose a node

$v_{\max}$ that contains the largest number of messages $K(v_{\max})$ that have not been broadcasted yet to all nodes in the network. The search for such a node is done via application of the broadcasting procedure $O(\log n)$ times, in principle, performing binary search in the set of nodes with $|K(v)| \leq 1$. One can prove that due to the property obtained on the conclusion of the first phase a number of iterations during the second phase is limited to $\sqrt{n}/\log n$, in total resulting in the complexity $O(n^{3/2}\log^2 n)$. This solution was further nicely polished by Xu in [47] who obtained $O(n^{3/2})$-time gossiping.

The first attempt to gossiping performed in directed ad-hoc radio networks with large labels can be found in [30]. The authors show how to utilise selectors with linear selectivity to reduce virtual degrees of nodes. This allowed to speed up the process of pushing messages through the networks more efficiently with a help of larger but also more selective (k, k, N)-selectors. They proposed a gossiping procedure with the time complexity $O(n^{5/3}\log^3 n)$. This result was further improved to $O(n^{4/3}\log^4 n)$ by Gąsieniec *et al.* in [34] with a help of *path selectors* formed of an appropriately balanced mix of selectors with different levels of selectivity. The path selectors proved to be more efficient than (k, k, N)-selectors in the process of pushing messages through the network with virtually reduced degrees of nodes. Since the only known lower bound on the time complexity of radio gossiping is $\Omega(n \log n)$ (gossiping in a star with n nodes requires this time in view of the size of selectors) the gap between the fastest currently known algorithm [34] and the lower bound constitutes one of the most challenging open problems in radio gossiping.

Randomised algorithms. Throughout the last decade there has been also a considerable interest in randomised radio gossiping. The first successful attempt was made by Chrobak *et al.* in [12] where they provided a nice dissemination mechanism based on the concept of the *distributed coupon collector problem* defined as follows. The network nodes stand for n bins and their messages serve as n coupons. Each coupon is available in at least k copies located in different bins, where $K(v)$ is the content of bin v. During each step the bins get open at random by choosing each bin independently with probability $1/n$. If exactly one bin v gets opened, all coupons from $K(v)$ are collected. Otherwise, a failure is experienced and no coupons are collected. On can prove the following lemma.

Lemma 1 ([12]). *For any $0 < \delta < 1$, repeating the random selection of a bin $(4n/k)\ln(n/\delta)$ times results in collection of all coupons with probability $\geq 1 - \delta$.*

Algorithm RANDOM-GOSSIP;
$\delta \leftarrow \epsilon/\log n$
for $i = 0, 1, \ldots, \log n - 1$ **do** $\{$*Stage* $i\}$
  repeat $(4n/2^i)\ln(n/\delta)$ **times**
    with probability $1/n$ **do**
      LTDBROADCAST$_v(2^{i+1})$

Note that procedure LTDBROADCAST$_v(2^{i+1})$ provides a dissemination mechanism based on selectors, as in [11], that allow to inform at least 2^{i+1} nodes

in time $O(2^{i+1}\log^2 n)$. Note also that one can adopt the following invariant. On the conclusion of each stage i, for $i = 0, .., \log n - 1$, each message is distributed to at least 2^{i+1} nodes. This is due to lemma 1 and the property of the procedure LtdBroadcast$_v(2^{i+1})$. Thus on the conclusion of the algorithm Random-Gossip all nodes get informed and the gossiping is accomplished. The probabilities are chosen such that with probability at least $1 - \epsilon$, algorithm Random-Gossip accomplishes gossiping in time $O(n \log^3 n \log(n/\epsilon))$. To obtain a Las Vegas type algorithm one can run Random-Gossip with $\epsilon = 1/n$ with the expected running time $O((1 - 1/n)n \log^4 n + (1/n)n^2) = O(n \log^4 n)$. Further work performed by Liu and Prabhakaran in [41] and Czumaj and Rytter in [15] lead to improvements $O(n \log^3 n)$ and $O(n \log^2 n)$ respectively.

Finally, note that gossiping in undirected ad-hoc radio networks is easier and it already has an almost linear solution. Gąsieniec *et al.* in [30] showed that if bidirectional links are available the time complexity of radio gossiping is $O(n \log^4 n)$ leaving only a polylogarithmic gap to be closed.

Small messages. Among other interesting models studied in the context of gossiping in ad-hoc radio networks we find work of Christersson *et al.* [10] on communication with *unit-size messages*, where it is assumed that the messages are limited to constant size, the connections are symmetric and the labels are drawn from the range $[1, .., O(n)]$. They show that in this restricted model gossiping can be accomplished deterministically in time close to $O(n^3/2)$. They also show that if messages are limited to size n^t, for any $0 < t < 1/2$, there is a gossiping algorithm with the time complexity $O(n^{2-t})$. Furthermore, by adopting known results on randomized broadcasting in symmetric ad-hoc radio networks they derive a randomized gossiping protocol with almost linear time complexity.

Max-degree Δ and diameter D. An alternative solution with the time complexity $O(D\Delta^2 \log^3 n)$ taking into account other network parameters such as the diameter D and the maximum in-degree Δ in the graph of connections can be obtained using work of Clementi *et al.* [13]. This result was further improved by Gąsieniec and Lingas in [29] where they proposed two alternative gossiping algorithms based on gradual construction of a map of connections in the network with the time complexity $O(nD^{1/2} \log^2 n)$ and $O(D\Delta^{3/2} \log^3 n)$.

3 Radio Networks with Known Topology

The situation changes dramatically if the network topology connections is known in advance. In this case one can schedule nodes' transmissions more efficiently minimising negative effect of collisions caused by simultaneous transmissions. In this case the main emphasis is on the design of efficient protocols with the time complexity linear or close to linear in the diameter D. Note, however, that due to the lower bound $\Omega(\log^2 n)$ from [1] the time complexity of any broadcasting and gossiping procedure must contain also a respective summand $O(\log^2 n)$. In [6], Chlamtac and Weinstein explain how to broadcast in time $O(\log^2 n)$ in bipartite

graphs of size n. In [33], Gąsieniec *et al.* define a notion of *minimal covering sets*. Let $B = (U \cup L, E)$, be a bipartite graph with two partitions of nodes U and L and edges in E with the endpoints in different partitions.

Definition 2 ([33]). *A minimal covering set (MCS) of nodes in L is any subset $U' \subseteq U$, s.t., all nodes in L have some neighbour in U' and removal of any node in U' would break this condition.*

Property 1. Each node in a minimal covering set has a unique neighbour in the other partition, otherwise such a node could be removed without breaking the covering property.

Assume that $|U| = m$ and $|L| = \mu$, where $m + \mu = n$, and that all nodes in U are already informed. We show how to use minimal covering sets to create a broadcast schedule between two partitions of a bipartite graph $B = (U \cup L, E)$ of length $O(\log^2 n)$. The goal is to inform all nodes in L. We show that one can select in time $O(m\mu)$ a subset of vertices in U that is able inform a fraction of $\frac{1}{2 \ln m}$ vertices in L in one round.

The construction starts with selection of an arbitrary minimal covering set U'. This can be done via removal of vertices (and edges adjacent to them) from U for as long as the covering property is not violated. At the end of this process every node in U' has at least one unique neighbour in L. Two cases occur.

Case 1: if $|U'| = m \geq \frac{\mu}{2 \ln m}$, due to the property of minimal covering sets simultaneous transmissions of all nodes in U' result in informing $\frac{\mu}{2 \ln m}$ nodes in L, otherwise

Case 2: we iterate the following process assuming that during consecutive iterations the cardinality of U' is i, starting with $i = m$. Now, if (a) each node in U' has at least $\frac{\mu}{2i \ln m}$ unique neighbours in L, then set U' can inform simultaneously $i \cdot \frac{\mu}{2i \ln m} = \frac{\mu}{2 \ln m}$ nodes in L. Otherwise, (b) there is a node in U' that has at most $\frac{\mu}{2i \log m}$ unique neighbours in L. Remove this node from U' and remove its unique neighbours from L.

The iteration process (Case 2) halts when, either case 2(a) occurs or exactly one node is left in U'. In the latter case we show that there is at least $\frac{\mu}{2}$ nodes in L, i.e., more than we need. And indeed, the total number of removed nodes from L is bounded by $\sum_{i=1}^{m} \frac{\mu}{2i \ln m} = \frac{\mu}{2 \ln m} \cdot \sum_{i=1}^{m} \frac{1}{i} \leq \frac{\mu}{2 \ln m} \cdot \ln m = \frac{\mu}{2}$. This concludes the proof that one can select a subset of nodes in U that is able inform a fraction of $\frac{1}{2 \ln m}$ vertices in L in one round.

The time complexity of the selection process is bounded by the number of edges in B, i.e., by $O(n\mu) = O(n^2)$. Since we ought to inform all nodes in L one need to construct further $O(\log^2 m) = O(\log^2 n)$ rounds of transmissions resulting in the total time complexity $O(n^2 \log n)$. We believe that our algorithm is an interesting alternative to the algorithm presented in [6].

Another important concept used extensively in communication algorithms in radio networks with known topology is a BFS tree with ranked nodes by *Strahler numbers* [45]. All leaves in the tree receive rank 0. The rank of every internal

node depends on the highest rank r among its children. If there is a unique child with the highest rank r the parent adopts the same rank. Otherwise the parent adopts the rank $r+1$. One can prove that the root receives the largest rank of size not greater than $\log n$.

3.1 Broadcasting

It is known that the problem of finding an optimal deterministic broadcasting schedule for any input graph is NP-hard, see [5]. The first efficient radio broadcasting schedule with the time complexity $O(D \log^2 n)$ is due to Chlamtac and Weinstein [6], where they utilise $O(\log^2 n)$-time broadcast procedure for bipartite graphs. This upper bound was later improved to $O(D \log n + \log^2 n)$ by Bar-Yehuda et al. in [2] who used a probabilistic argument. An explicit deterministic construction can be found in [39].

In [25] Gaber and Mansour proposed partitioning of the graph of connections into super-levels and a system (graph structure) of clusters, s.t.,

1. each cluster has a relatively small diameter,
2. the union of the clusters covers the super-level, and
3. the clusters graph can be colored by O(log n) colors.

The clusters graph is obtained by treating each cluster as a node, and introducing an edge between two nodes if in the original graph there is some edge that connects nodes from the corresponding clusters or if they share a neighbour. Clusters in different super-layers are spanned by a ranked BFS tree. During broadcasting process *fast transmissions* are performed along downwards paths in the tree cutting through clusters with the same rank. Other (*slow*) transmissions, including local transmissions within clusters, are implemented with a help of the $O(\log^2 n)$-time broadcast mechanism from [6]. This new construction supported by the randomized scheme from [2] resulted in the broadcasting schedule of length $O(D + log^5 n)$, and by the deterministic scheme from [6] in fully constructive $O(D + log^6)$-time broadcasting schedule. More recently, Elkin and Kortsarz in [21] proposed an improved $O(D + \log^4 n)$-time version of the broadcasting schedule from [25] and they showed how to broadcast messages in planar graphs in time $O(D + \log^3 n)$.

In more recent work [31] Gąsieniec *et al.* proposed a more direct use of ranked BFS spanning trees built directly on network nodes rather than clusters, resulting in a deterministic broadcasting schedule of length $D + O(log^3 n)$ and its randomized counterpart of length $D + O(log^2 n)$. They also proved that in planar graphs one can broadcast messages in time $3D$, i.e., in time independent from n. An alternative deterministic construction of $O(D + \log^2 n)$-time broadcast schedule can be found in [39]. Finally we mention two very recent results on k-shot broadcast schedules in which network nodes are allowed to transmit at most k times. In [27] the authors present, e.g., an almost optimal 1-shot broadcast schedule of length $D + O(\sqrt{n} \log n)$. In [26] Galcik *et al.* consider radio broadcasting model in which interference range of nodes is likely to exceed their

transmission ranges. They show, using an extension of the system of clusters from [25], how to compute a 1-shot broadcasting schedule of length at most $4 \cdot D_T + O(\Delta \cdot d_I \cdot \log^4 n)$ for networks with arbitrary topology, where d_I is the *interference distance* parameter.

3.2 Gossiping

A discussion on radio gossiping in known arbitrary graphs was initiated by Gąsieniec *et al.* in their seminal paper [33]. They proved, e.g., that in any network of size n gossiping is feasible in time at most n. The proof is based on the concept of a *2-vertex reduction principle*.

Lemma 2 (2-Vertex Reduction Principle [33]). *In any undirected graph $G = (V, E)$ with a radius greater than 1 there exist four distinct nodes v, v', w and w', such that, edges $(v, v'), (w, w') \in E$ and $(v, w')(w, v') \notin E$ and removal of both v and w does not disconnect the remaining part of the graph.*

They also provided the lower bound $\lfloor \log(n-1) \rfloor + 2$ for gossiping in any radio network and showed that this bound can be matched from above, in specific graph structures, for a large fraction of all integer values of n. For all other values the upper bound $\lceil \log(n-1) \rceil + 2$ is established. They also proposed a modification of the system of clusters from [25] to produce gossiping schedules of length $O(D + \sqrt[i+2]{D}\Delta \log^{i+1} n)$, for any network with the diameter $D = \Omega(\log^{i+4} n)$, where i is an arbitrary integer constant $i \geq 0$.

This result was pruned later by Gąsieniec *et al.* in [31] where they proposed deterministic construction of a $O(D + \Delta \log n)$-time schedule based on the concept of ranked BFS trees and efficient $O(\Delta)$-time transfer (based on property of minimum covering sets) of gossip messages between partitions of a bipartite graph of the max-degree Δ. This is the best currently known upper bound on gossiping in radio networks with known topology.

Small messages. An interesting variant of gossiping in radio networks was studied by Gąsieniec and Potapov in [32], where they assume that messages exchanged between neighbouring nodes are limited to constant size. Under this assumption they established the time complexity for gossiping schedules on lines and rings of size n with lengths $3n + \Theta(1)$ and $2n + \Theta(1)$ respectively. They also proposed linear time schedules for trees and proved bounds $\Omega(n \log n)$ and $O(n \log^2 n)$ on the time complexity of gossiping in arbitrary graphs in this variant of radio networks. The upper bound was later improved to $O(n \log n)$ by Manne and Xin in [42] with a help of a probabilistic argument.

M2M multicast. The *M2M (multi-to-multi) multicast* problem, see [28], is a close relative of gossiping that seats somewhere on the border of ad-hoc networks and networks with known topology. In M2M problem it is assumed that k out of n nodes are woken up at some particular round and they are asked to communicate

with one another as fast as possible. While the radio network topology is known to all nodes, it is assumed that no node is aware of the location of $k-1$ other participants. The authors provide a distributed deterministic algorithm for the M2M multicast problem using an extension of the system of clusters from [25]. They show that if the maximum distance between any two out of k participants is d all nodes can find each other in time $O(d \log^2 n + k \log^4 n)$.

4 Conclusion

In this paper we gave a short introduction to radio gossiping and we discussed several combinatorial concepts used in the context of developed efficient algorithmic solutions. There are, however, a numerous variants of efficient radio gossiping methods that are not listed here. These include, e.g., gossiping in random graphs with limited number of transmissions at each node [3], variants of $O(D)$-time gossiping in *random geometric ad-hoc networks* [16], efficient randomised gossiping in *geometric sensor networks* [44,24], and many, many others. This short survey focuses mainly on core developments in radio gossiping in networks with arbitrary topology.

Among the most interesting open problems in the field is efficient construction of (k, m, n)-selectors. In fact, surprisingly, we still don't know whether construction of selectors of optimal size is NP-hard or not. On one hand, we have certain exponential constructions of optimal selectors [17] and on the other we know how to generate their rough approximations [36,9]. Further study on more satisfactory trade-offs would be highly appreciated. Another important objective is to establish tighter bounds for deterministic and randomised gossiping in ad-hoc (un)directed radio networks. Similarly, better understanding of gossiping in radio networks with known topology is still needed. For example, is it possible to design a good approximate gossiping schedule for any specific input graph G? Finally, not much if nothing has been done in alternative models for radio communication including SINR model [35].

References

1. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A lower bound for radio broadcast. Journal of Computer and System Sciences 43, 290–298 (1991)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time complexity of broadcast in radio networks: an exponential gap between determinism and randomization. Journal of Computer and System Sciences 45, 104–126 (1992)
3. Berenbrink, P., Cooper, C., Hu, Z.: Energy efficient randomised communication in unknown AdHoc networks. In: Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2007, pp. 250–259 (2007)
4. Cheng, Y., Du, D.Z.: New constructions of one- and two-stage pooling designs. Journal on Computational Biology 15, 195–205 (2008)

5. Chlamtac, I., Kutten, S.: On broadcasting in radio networks - problem analysis and protocol design. IEEE Transactions on Communications 33, 1240–1246 (1985)
6. Chlamtac, I., Weinstein, O.: The wave expansion approach to broadcasting in multihop radio networks. IEEE Transactions on Communications 39(9), 426–433 (1991)
7. Chlebus, B.S., Gąsieniec, L., Östlin, A., Robson, J.M.: Deterministic Radio Broadcasting. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 717–728. Springer, Heidelberg (2000)
8. Chlebus, B.S., Gąsieniec, L., Gibbons, A., Pelc, A., Rytter, W.: Deterministic broadcasting in unknown radio networks. Distributed Computing 15(1), 27–38 (2002)
9. Chlebus, B.S., Kowalski, D.R.: Almost Optimal Explicit Selectors. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 270–280. Springer, Heidelberg (2005)
10. Christersson, M., Gąsieniec, L., Lingas, A.: Gossiping with Bounded Size Messages in ad hoc Radio Networks. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 377–389. Springer, Heidelberg (2002)
11. Chrobak, M., Gąsieniec, L., Rytter, W.: Fast broadcasting and gossiping in radio networks. Journal on Algorithms 43(2), 177–189 (2002)
12. Chrobak, M., Gąsieniec, L., Rytter, W.: A randomized algorithm for gossiping in radio networks. Networks 43(2), 119–124 (2004)
13. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: Proc. 12th Annual ACM-SIAM symposium on Discrete algorithms, SODA 2001, pp. 709–718 (2001)
14. Clementi, A.E.F., Monti, A., Silvestri, R.: Distributed broadcasting in radio networks of unknown topology. Theoretical Computer Science 302, 337–364 (2003)
15. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. Journal on Algorithms 60(2), 115–143 (2006)
16. Czumaj, A., Wang, X.: Fast Message Dissemination in Random Geometric Ad-Hoc Radio Networks. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 220–231. Springer, Heidelberg (2007)
17. De Bonis, A., Gąsieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. SIAM Journal on Computing 34(5), 1253–1270 (2005)
18. De Marco, G.: Distributed broadcast in unknown radio networks. In: Proc. of 19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, pp. 208–217 (2008)
19. De Marco, G., Pelc, A.: Faster broadcasting in unknown radio networks. Information Processing Letters 79, 53–56 (2001)
20. Du, D.Z., Hwang, F.K.: Combinatorial Group Testing and Its Applications. World Scientific, River Edge (2000)
21. Elkin, M., Kortsarz, G.: An improved algorithm for radio broadcast. ACM Transactions on Algorithms 3(1), 1–21 (2007)
22. Eppstein, D., Goodrich, M.T., Hirschberg, D.S.: Improved combinatorial group testing algorithms for real-world problem sizes. SIAM Journal on Computing 36, 1360–1375 (2007)
23. Erdös, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of r others. Israel Journal of Mathematics 51, 75–89 (1985)

24. Farach-Colton, M., Mosteiro, M.A.: Sensor Network Gossiping or How to Break the Broadcast Lower Bound. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 232–243. Springer, Heidelberg (2007)
25. Gaber, I., Mansour, Y.: Centralized broadcast in multihop radio networks. Journal of Algorithms 46(1), 1–20 (2003)
26. Galčík, F., Gąsieniec, L., Lingas, A.: Efficient broadcasting in known topology radio networks with long-range interference. In: Proc. 28th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2009, pp. 230–239 (2009)
27. Gąsieniec, L., Kantor, E., Kowalski, D.R., Peleg, D., Su, C.: Time efficient k-shot broadcasting in known topology radio networks. Distributed Computing 21(2), 117–127 (2008)
28. Gąsieniec, L., Kranakis, E., Pelc, A., Xin, Q.: Deterministic M2M multicast in radio networks. Theoretical Computer Science 362(1-3), 196–206 (2006)
29. Gąsieniec, L., Lingas, A.: On adaptive deterministic gossiping in ad hoc radio network. In: Proc. 13th Annual ACM-SIAM symposium on Discrete Algorithms, SODA 2002, pp. 689–690 (2002)
30. Gąsieniec, L., Pagourtzis, A., Potapov, I.: Deterministic Communication in Radio Networks with Large Labels. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 512–524. Springer, Heidelberg (2002)
31. Gąsieniec, L., Peleg, D., Xin, Q.: Faster communication in known topology radio networks. Distributed Computing 19(4), 289–300 (2007)
32. Gąsieniec, L., Potapov, I.: Gossiping with Unit Messages in Known Radio Networks. In: IFIP TCS 2002, pp. 193–205 (2002)
33. Gąsieniec, L., Potapov, I., Xin, Q.: Time efficient centralized gossiping in radio networks. Theoretical Computer Science 383(1), 45–58 (2007)
34. Gąsieniec, L., Radzik, T., Xin, Q.: Faster Deterministic Gossiping in Directed Ad Hoc Radio Networks. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 397–407. Springer, Heidelberg (2004)
35. Gupta, P., Kumar, P.R.: The Capacity of Wireless Networks. IEEE Transactions on Information Theory 46(2), 388–404 (2000)
36. Indyk, P.: Explicit constructions of selectors and related combinatorial structures, with applications. In: Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002, pp. 697–704 (2002)
37. Kautz, W.H., Singleton, R.R.: Nonrandom binary superimposed codes. IEEE Transactions on Information Theory 10, 363–377 (1964)
38. Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad hoc radio networks. Distributed Computing 18(1), 43–57 (2005)
39. Kowalski, D.R., Pelc, A.: Optimal Deterministic Broadcasting in Known Topology Radio Networks. Distributed Computing 19(3), 185–195 (2007)
40. Kushilevitz, E., Mansour, Y.: An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks. SIAM Journal on Computing 27, 702–712 (1998)
41. Liu, D., Prabhakaran, M.: On Randomized Broadcasting and Gossiping in Radio Networks. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 340–349. Springer, Heidelberg (2002)
42. Xin, Q., Manne, F.: Optimal Gossiping with Unit Size Messages in Known Topology Radio Networks. In: Erlebach, T. (ed.) CAAN 2006. LNCS, vol. 4235, pp. 125–134. Springer, Heidelberg (2006)

43. Peleg, D.: Deterministic radio broadcast with no topological knowledge (2000) (Manuscript)
44. Ravelomanana, V.: Optimal initialization and gossiping algorithms for random radio networks. IEEE Transactions on Parallel Distributed Systems 18(1), 17–28 (2007)
45. Strahler, A.N.: Hypsometric (area-altitude) analysis of erosional topology. Bulletin of Geological Society of America 63, 1117–1142 (1952)
46. Vishkin, U.: Deterministic sampling - A new technique for fast pattern matching. SIAM Journal on Computing 20, 22–40 (1991)
47. Xu, Y.: An $O(n^{3/2})$ deterministic algorithm for radio networks. Algorithmica 36(1), 93–96 (2003)

Regular Register: An Implementation in a Churn Prone Environment

Roberto Baldoni[1], Silvia Bonomi[1], and Michel Raynal[2]

[1] Universitá La Sapienza, Via Ariosto 25, I-00185 Roma, Italy
{baldoni,bonomi}@dis.uniroma1.it
[2] IRISA, Université de Rennes, Campus de Beaulieu, F-35042 Rennes, France
raynal@irisa.fr

Abstract. Due to their capability to hide the complexity generated by the messages exchanged between processes, shared objects are one of the main abstractions provided to the developers of distributed applications. Among all the shared objects, the register object is fundamental. Several protocols have been proposed to build fault resilient registers on top of message-passing system, but, unfortunately, failure are not the only challenge in modern distributed systems. New issues arise from the dynamicity introduced in the system by the continuous arrival and departure of nodes (*churn* phenomenon). This paper addresses the construction of a single writer/multiple readers regular register in a distributed system affected by the continuous arrival/departure of participants. In particular, a general protocol implementing a regular register is proposed and feasibility conditions on the arrival and departure of the processes are given. Interestingly, the protocol is proved correct under the assumption that the constraint on the churn is satisfied.

1 Introduction

Context and motivation. Dealing with failure has been one of the main challenge in defining abstractions (shared memory, communication, agreement, etc.) able to work in a distributed system. Many protocols have been designed to allow such abstractions to behave correctly despite asynchrony and failures. In nearly all the cases, the system is always "well defined" in the sense that the whole set of participating processes is finite and known in advance by each process. The system composition is modified only when a process crashes.

A new challenge is emerging due to the advent of new classes of applications and technologies. More specifically, "modern" distributed systems are characterized by the unpredictability of the system composition that is caused by the arrival of new processes that become new members of the system or the departure of participating processes. As a consequence of such an unpredictability, distributed computing abstraction have to deal not only with asynchrony and failures, but also with the system dynamicity dimension. Hence, the abstractions and protocols implementing them have to be reconsidered to take into account

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 15–29, 2010.

this new "adversary" setting. Dynamicity makes abstractions more difficult to understand and master than in classical static distributed systems where the set of processes is fixed once for all. The *churn* notion has been introduced to capture this dynamicity feature. It is a system parameter the aim of which is to make tractable systems whose composition evolves with time (e.g., [9,14,16]).

Although numerous protocols have been designed for dynamic distributed message-passing systems, few papers (such as [1,2,19]) strive to present models suited to such systems, and extremely few dynamic protocols have been proved correct. While up to now the most common approach used to address dynamic systems is mainly experimental, we have presented in [3] a protocol that constructs a register in a dynamic system the churn of which remains always constant and is know by each process.

Contribution and roadmap. This paper extends our previous work in a setting where churn and the number of processes in the system may vary. In [3], we indeed characterize a model of churn in which the number of processes n in the system is always constant (this means at any time the same number of processes join and leave the system). Upon this model, we have proved that: (i) a regular register cannot be build in an asynchronous system, (ii) a regular register can be implemented in an eventually synchronous distributed system if at any time there is a number of active processes greather than $\lceil n/2 \rceil$ and (iii) a regular register can be implemented in a synchronous distributed system if at any time there is at least one active process and the percentage of processes that can change at any time is less than a constant depending on the protocol implementation. In this paper, we first introduce a very generic model of churn, based on a joining and a departure distributions, in which the number of processes in the system remains, at any time, in a given range. Once the range is set, this turns out in contraints on joining and departure distributions. Interestingly, this churn model is able to capture the infinite arrival model presented in [20] and the constant churn model presented in [9,3] as specific cases. Secondly, we take the implementation of a single writer/multiple reader regular register in a synchronous system shown in [3] and compute the additional constraints imposed by such implementation on the churn (i.e. on the joining and on the departure distributions) in order that the regular register be correct.

The paper introduces the system and the churn model in Section 2 and Section 3, respectively. Section 4 presents the notion of a regular register and Section 5 reports, for the sake of completeness, the implementation of such a register as presented in [3]. Section 6 proves the constraints that have to be imposed on the departure and the join distributions in order the register be correct. A related work and a concluding remark sections conclude the paper.

2 System Model

The distributed system is composed, at each time, by a bounded number of processes that communicate by exchanging messages. Processes are uniquely

identified (with their indexes) and they may join and leave the system at any point in time.

The system is synchronous in the following sense: the processing times of local computations are negligible with respect to communication delays, so they are assumed to be equal to 0. Contrarily, messages take time to travel to their destination processes. Moreover we assume that processes can access a global clock[1].

We assume that there exists an underling protocol, that keeps processes connected each other. This protocol is implemented at the connectivity layer (the layer at the bottom of Figure 1).

2.1 Distributed Computation

A distributed computation is formed, at each instant of time, by a subset of processes of the distributed system. A process p, belonging to the system, that wants to participate to the distributed computation has to execute the join() operation. Such operation, invoked at some time t, is not instantaneous: it consumes time. But, from time t, the process p can receive and process messages sent by any other process that belongs to the system and that participate to the computation. Processes participating to the distributed computation implements a regular register abstraction.

A process leaves the computation in an implicit way. When it does, it leaves the computation forever and does not longer send messages. From a practical point of view, if a process wants to re-enter the system, it has to enter it as a new process (i.e., with a new name).

We assume that a process does not crash during the distributed computation (i.e. it does not crash from the time it joins the system until it leaves).

In order to formalize the set of processes that participate actively to the computation we give the following definition.

Definition 1. *A process is* active *from the time it returns from the* join() *operation until the time it leaves the system. $A(t)$ denotes the set of processes that are active at time t, while $A([t_1, t_2])$ denotes the set of processes that are active during the interval $[t_1, t_2]$.*

2.2 Communication Primitives

Two communication primitives are used by processes belonging to the distributed computation to communicate: point-to-point and broadcast communication as shown in Figure 1.

Point-to-point communication. This primitive allows a process p_i to send a message to another process p_j as soon as p_i knows that p_j has joined the computation. The network is reliable in the sense that it does not loose, create

[1] The global clock is for ease of presentation. As we are in a synchronous system, this global clock can be implemented by synchronized local clocks.

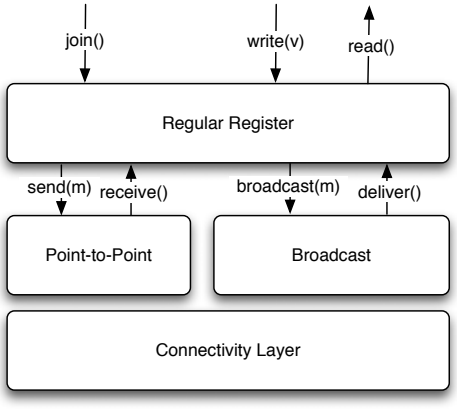


Fig. 1. System architecture

or modify messages. Moreover, the synchrony assumption guarantees that if p_i invokes "send m to p_j" at time t, then p_j receives that message by time $t + \delta'$ (if it has not left the system by that time). In that case, the message is said to be "sent" and "received".

Broadcast. Processes participating to the distributed computation are equipped with an appropriate broadcast communication sub-system that provides the processes with two operations, denoted broadcast() and deliver(). The former allows a process to send a message to all the processes in the distributed system, while the latter allows a process to deliver a message. Consequently, we say that such a message is "broadcast" and "delivered". These operations satisfy the following property.

– Timely delivery: Let t be the time at which a process p belonging to the distributed computation invokes broadcast(m). There is a constant δ ($\delta \geq \delta'$) (known by the processes) such that if p does not leave the system by time $t+\delta$, then all the processes that are in the system at time t and do not leave by time $t + \delta$, deliver m by time $t + \delta$.

Such a pair of broadcast operations has first been formalized in [6] in the context of systems where process can commit crash failures. It has been extended to the context of dynamic systems in [5].

3 Churn Model

3.1 Definitions

The dynamicity due to the continuous join and leave of nodes in the system is a phenomenon identified under the name *churn*. This section introduces a general model able to characterize such dynamicity. The model proposed here is based mainly on the definition of two distributions (i) the join distribution $\lambda(t)$ that

defines the join of new processes to the system with respect to time and (ii) the leave distribution $\mu(t)$ that defines the leave of processes from the system with respect to time. Such distributions are discrete function of time:

Definition 2. (Join distribution) *The* join distribution $\lambda(t)$ *is a discrete time function that returns the number of processes that invoke the join operation at time* t*.*

Definition 3. (Leave distribution) *The* leave distribution $\mu(t)$ *is a discrete time function that returns the number of processes that have left the system at time* t*.*

Let t_0 be the starting time of the system. We assume that at time t_0 no process joins or leaves the system then $\lambda(t_0) = 0$ and $\mu(t_0) = 0$ therefore we can say that in t_0 the system is composed by a set Π_0 of processes and the size of the system is n_0 (i.e. $|\Pi_0| = n_0$). Moreover, for any time $t < t_0$ we say that $\lambda(t) = \mu(t) = 0$.

Note that a static system is a system characterized by a join distribution and a leave distribution that are always equal to 0 for any time t (i.e. $\lambda(t) = \mu(t) = 0$ $\forall t$).

As soon as the dynamicity introduced by the joins and the leaves appears in the system, it is possible to observe that the size of network and its composition can change.

Let $N(t)$ be a discrete time function that returns the number of processes inside the system at time t. Depending on the values of $\lambda(t)$ and $\mu(t)$ we have:

Definition 4. (Node distribution) *Let* n_0 *be the number of processes in the system at start time* t_0*.* $N(t)$ *is the number of processes within the system at time* t *for every* $t \geq t_0$ *(i.e.* $N(t) = N(t-1) + \lambda(t) - \mu(t)$*, with* $N(t_0) = n_0$*).*

Composition of the System. The variation of the network size is not the only effect generated by the dynamicity introduced by the join and leave of nodes. The composition of network is also affected. As an example consider a network of size n, and a join distribution and a leave distribution that are constant and equal (i.e $\lambda(t_i) = \mu(t_i) = c, \forall i$ with $c \leq n \in \mathbb{N}$). In such a system, the dynamicity does not affect the network size, it affect only its composition. In order to capture this dynamicity effect, we define the *leave percentage* of the network in a given interval.

Definition 5. (LeavePercentage) *Let* n_0 *be the number of processes in the system at start time* t_0*, let* $\lambda(t)$ *the join distribution and* $\mu(t)$ *the leave distribution of the system. Let* $\Delta t = [t, t + \Delta]$ *be a time interval.* $L(\Delta t)$ *is the percentage of processes that have been refreshed (i.e., that have left the system) during the interval* Δt*, i.e.* $L(\Delta t) = \frac{\sum_{\tau=t}^{t+\Delta} \mu(\tau)}{N(t-1)}$ *(with* $N(t_0) = n_0$*).*

This metric allows to state a simple lemma on the composition of the system.

Lemma 1. *Let* $\Delta t = [t, t + \Delta]$ *be a time interval. If there exists a process* p *that is in the system at time* t*and does not leave for all the interval* Δt *then* $L(\Delta t) < 1$.

3.2 Churn Constraint on the Upper and Lower Bounds on the Network Size

Based on the previous definitions, this section derives the constraint that a join distribution and a leave distribution have to satisfy in order the network size remains in a given interval. Let n_0 be the number of processes inside the system at the start time t_0 and k_1, k_2 be two positive integers. The aim is to model a network affected by the dynamicity whose effect is the variation of the system size in an interval ΔN that is defined by the upper bound $n_0 + k_2$ and the lower bound $n_0 - k_1$ (i.e. $\Delta N = [n_0 - k_1, n_0 + k_2]$).

Lemma 2. *Let k_1 and k_2 be two integers such that $k_1, k_2 \geq 0$ and let n_0 be the number of processes in the system at the starting time t_0. Given the join and the leave distribution $\lambda(t)$ and $\mu(t)$, the node distribution $N(t)$ is always comprised inside the interval $\Delta N = [n_0 - k_1, n_0 + k_2]$ if and only if:*

(c1) $\sum_{\tau=t_0}^{t} \mu(\tau) \leq \sum_{\tau=t_0}^{t} \lambda(\tau) + k_1 \ \forall t$,
(c2) $\sum_{\tau=t_0}^{t} \mu(\tau) \geq \sum_{\tau=t_0}^{t} \lambda(\tau) - k_2 \ \forall t$.

Proof. Due to Definition 4, we have that for each time t, $N(t) = N(t-1) + \lambda(t) - \mu(t)$ and iterating, we can express the node distribution as function of n_0 as $N(t) = n_0 + \sum_{\tau=t_0}^{t} \lambda(\tau) - \sum_{\tau=t_0}^{t} \mu(\tau)$. Constraining $N(t)$ to be in the interval ΔN we obtain, for every t, for the lower bound

$$\begin{array}{r} N(t) \geq n_0 - k_1 \\ n_0 + \sum_{\tau=t_0}^{t} \lambda(\tau) - \sum_{\tau=t_0}^{t} \mu(\tau) \geq n_0 - k_1 \\ \sum_{\tau=t_0}^{t} \mu(\tau) \leq \sum_{\tau=t_0}^{t} \lambda(\tau) + k_1, \end{array}$$

and we obtain, for every t, for the upper bound

$$\begin{array}{r} N(t) \leq n_0 + k_2 \\ n_0 + \sum_{\tau=t_0}^{t} \lambda(\tau) - \sum_{\tau=t_0}^{t} \mu(\tau) \leq n_0 + k_2 \\ \sum_{\tau=t_0}^{t} \mu(\tau) \geq \sum_{\tau=t_0}^{t} \lambda(\tau) - k_2. \end{array}$$

□*Lemma 2*

4 Regular Register

Regular register. A register is a shared object by a set of processes. Such an object provides processes with two operation, namely read() and write(), that allow them to read the value contained in the object or to modify such a value. Depending on the semantic of the operations, several types of register have been defined by Lamport [12]. A regular register can have any number of writers and any number of readers. The writes appear as if they were executed sequentially, this sequence complying with their real time order (i.e., if two writes w_1 and w_2 are concurrent, they can appear in any order, but if w_1 terminates before w_2 starts, w_1 has to appear as being executed before w_2). As far as a read operation is concerned we have the following. If no write operation is concurrent with a

read operation, that read operation returns the current value kept in the register. Otherwise, the read operation returns any value written by a concurrent write operation or the last value of the register before these concurrent writes.

Here we focus our attention on a one-writer/multi-reader *regular register*[2].

Specification for a dynamic system. The notion of a regular register has to be adapted to dynamic systems. We consider that a protocol implements a regular register in a dynamic system if the following properties are satisfied.

- Liveness: If a process invokes a read or a write operation and does not leave the system, it eventually returns from that operation.
- Safety: A read operation returns the last value written before the read invocation, or a value written by a write operation concurrent with it.

Moreover, it is assumed that a process invokes the read or write operation only after it has returned from its join() invocation [3].

5 A General Protocol for Synchronous Dynamic Systems

In this section we present, for completeness, the protocol presented in [3] implementing a single writer/multireader regular register.

The principle that underlies the design of the protocol is to have *fast reads* operations: a process willing to read has to do it locally. From an operational point of view, this means that a read is not allowed to use a wait() statement, or to send messages and wait for associated responses. Hence, albeit the proposed protocol works in all cases, it is targeted for applications where the number of reads outperforms the number of writes.

Local variables at a process p_i. Each process p_i has the following local variables.

- Two variables denoted $register_i$ and sn_i; $register_i$ contains the local copy of the regular register, while sn_i is the associated sequence number.
- A boolean $active_i$, initialized to *false*, that is switched to *true* just after p_i has joined the system.
- Two set variables, denoted $replies_i$ and $reply\_to_i$, that are used during the period during which p_i joins the system. The local variable $replies_i$ contains the 3-uples $< id, value, sn >$ that p_i has received from other processes during its join period, while $reply\_to_i$ contains the processes that are joining the system concurrently with p_i (as far as p_i knows).

Initially, n processes compose the system. The local variables of each of these processes p_k are such that $register_k$ contains the initial value of the regular register[3], $sn_k = 0$, $active_k = true$, and $replies_k = reply\_to_k = \emptyset$.

[2] Actually, the protocol proposed in Section 5 works for any number of writers as long as the writes are not concurrent. Considering a single writer makes the exposition easier.

[3] Without loss of generality, we assume that at the beginning every process p_k has in its variable $register_k$ the value 0.

```
operation join(i):
(01) register_i ← ⊥; sn_i ← −1; active_i ← false; replies_i ← ∅; reply_to_i ← ∅;
(02) wait(δ);
(03) if (register_i = ⊥) then
(04)     replies_i ← ∅;
(05)     broadcast INQUIRY(i);
(06)     wait(2δ);
(07)     let < id, val, sn >∈ replies_i such that (∀ < −, −, sn' >∈ replies_i : sn ≥ sn');
(08)     if (sn > sn_i) then sn_i ← sn; register_i ← val end if
(09) end if;
(10) active_i ← true;
(11) for each j ∈ reply_to_i do send REPLY (< i, register_i, sn_i >) to p_j;
(12) return(ok).
------------------------------------------------------------------------
(13) when INQUIRY(j) is delivered:
(14)       if (active_i) then send REPLY (< i, register_i, sn_i >) to p_j
(15)                     else reply_to_i ← reply_to_i ∪ {j}
(16)       end if.

(17) when REPLY(< j, value, sn >) is received: replies_i ← replies_i ∪ {< j, value, sn >}.
```

Fig. 2. The join() protocol for a synchronous system (code for p_i)

The join() operation. When a process p_i enters the system, it first invokes the join operation. The algorithm implementing that operation, described in Figure 2, involves all the processes that are currently present (be them active or not).

First p_i initializes its local variables (line 01), and waits for a period of δ time units (line 02); this waiting period is explained later. If $register_i$ has not been updated during this waiting period (line 03), p_i broadcasts (with the broadcast() operation) an INQUIRY(i) message to the processes that are in the system (line 05) and waits for 2δ time units, i.e., the maximum round trip delay (line 06)[4]. When this period terminates, p_i updates its local variables $register_i$ and sn_i to the most up-to-date values it has received (lines 07-08). Then, p_i becomes active (line 10), which means that it can answer the inquiries it has received from other processes, and does it if $reply\_to \neq \emptyset$ (line 11). Finally, p_i returns ok to indicate the end of the join() operation (line 12).

When a process p_i receives a message INQUIRY(j), it answers p_j by sending back a REPLY$(< i, register_i, sn_i >)$ message containing its local variable if it is active (line 14). Otherwise, p_i postpones its answer until it becomes active (line 15 and lines 10-11). Finally, when p_i receives a message REPLY$(< j, value, sn >)$ from a process p_j it adds the corresponding 3-uple to its set $replies_i$ (line 17).

[4] The statement wait(2δ) can be replaced by wait$(\delta+\delta')$, which provides a more efficient join operation; δ is the upper bound for the dissemination of the message sent by the reliable broadcast that is a one-to-many communication primitive, while δ' is the upper bound for a response that is sent to a process whose id is known, using a one-to-one communication primitive. So, wait(δ) is related to the broadcast, while wait(δ') is related to point-to-point communication. We use the wait(2δ) statement to make the presentation easier.

operation read(): return($register_i$). % issued by any process p_i %

operation write(v): % issued only by the writer p_w%
(01) $sn_w \leftarrow sn_w + 1$; $register_i \leftarrow v$ broadcast WRITE(v, sn_w);
(02) wait(δ); return(ok).

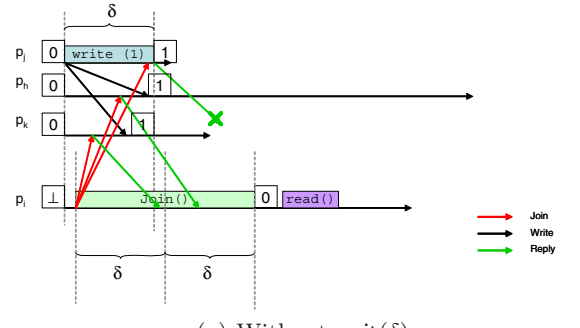
(03) **when** WRITE($< val, sn >$) is delivered: % at any process p_i %
(04) **if** ($sn > sn_i$) **then** $register_i \leftarrow val$; $sn_i \leftarrow sn$ **end if**.

Fig. 3. The read() and write() protocols for a synchronous system

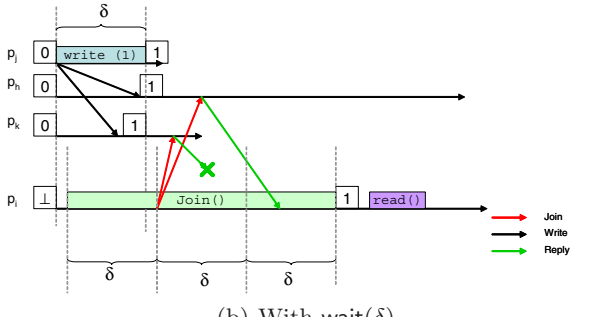
The read() and write(v) operations. The algorithms for the read and write operations associated with the regular register are described in Figure 3. The read is purely local (i.e., fast): it consists in returning the current value of the local variable $register_i$.

The write consists in disseminating the new value v (together with its sequence number) to all the processes that are currently in the system (line 01). In order to guarantee the correct delivery of that value, the writer is required to wait for δ time units before terminating the write operation (line 02).

Why the wait(δ) statement at line 02 of the join() operation? To motivate the wait(δ) statement at line 02, let us consider the execution of the join() operation depicted in Figure 4(a). At time t, the processes p_j, p_h and p_k are the



(a) Without wait(δ)



(b) With wait(δ)

Fig. 4. Why wait(δ) is required

three processes composing the system, and p_j is the writer. Moreover, the process p_i executes join() just after t. The value of the copies of the regular register is 0 (square on the left of p_j, p_h and p_k), while $register_i = \perp$ (square on its left). The 'timely delivery" property of the broadcast invoked by the writer p_j ensures that p_j and p_k deliver the new value $v = 1$ by $t+\delta$. But, as it entered the system after t, there is no such a guarantee for p_i. Hence, if p_i does not execute the wait(δ) statement at line 02, its execution of the lines 03-09 can provide it with the previous value of the regular register, namely 0. If after obtaining 0, p_i issues another read it obtains again 0, while it should obtain the new value $v = 1$ (because 1 is the last value written and there is no write concurrent with this second read issued by p_i).

The execution depicted in Figure 4(b) shows that this incorrect scenario cannot occur if p_i is forced to wait for δ time units before inquiring to obtain the last value of the regular register.

6 Churn Constraints Imposed by the Protocol for Register Correctness

Correctness of the register has to be proved by showing liveness and safety of the implementation. Liveness follows trivially by the fact that the wait() statement terminates in join() and write() operations. Concerning safety, as proved in [3], the safety of the regular register specification is satisfied by the existence of at least one active process for every period of 3δ time. This section formally proves additional constraints (for a system whose size is limited within an interval) the joining and departure distributions have to satisfy in order to have at least one active process in the system at any time.

Lemma 3. *Let t_0 be the starting time of the system and let n_0 be the number of processes inside the system at time t_0. Let $\sum_{\tau=t-3\delta+1}^{t} \mu(\tau) < N(t-3\delta)\ \forall t$, then $|A(t)| > 0\ \forall t$.*

Proof. At time t_0 the system is composed of n_0 processes and each of these process maintains the value of the register so we have that $|A(t_0)| = n_0$. By the definition of the join and leave distribution, we have that at time $t_0 + 1$ $\lambda(t_0 + 1)$ processes start the join operation and $\mu(t_0 + 1)$ processes leave the system then $|A(t_0 + 1)| = n_0 - \mu(t_0 + 1)$. At time $t_0 + 2$ we have that $\lambda(t_0 + 2)$ more processes invoke the join and $\mu(t_0 + 2)$ more processes leave the system. In the worse case, all the processes leaving the system are active hence $|A(t_0+2)| \geq n_0 - \mu(t_0+1) - \mu(t_0+2)$. To be completed, a join operation needs, in the worse case, 3δ time then until time $t_0 + 3\delta$ the number of the active processes always decrease and $|A(t_0 + 3\delta)| \geq n_0 - \sum_{\tau=t_0+1}^{t_0+3\delta} \mu(\tau)$. At time $t_0 + 3\delta + 1$, the joins invoked at time $t_0 + 1$ terminate and such joining processes become active so $|A(t_0+3\delta+1)| \geq n_0 - \sum_{\tau=t_0+1}^{t_0+3\delta+1} \mu(\tau) + \lambda(t_0+1)$. At time $t_0+3\delta+2$, also the join operations invoked at time t_0+2 terminate and $\lambda(t_0+2)$ more processes become active. By iteration, for a given t, $|A(t)| \geq n_0 - \sum_{\tau=t_0+1}^{t} \mu(\tau) + \sum_{\tau'=t_0+1}^{t-3\delta} \lambda(\tau')$

$= N(t-3\delta) - \sum_{\tau=t-3\delta+1}^{t} \mu(\tau)$. Moreover, as $\sum_{\tau=t-3\delta+1}^{t} \mu(\tau) < N(t-3\delta)\ \forall t$, we have that $|A(t)| > 0$ for any time t. $\square_{Lemma\ 3}$

Lemma 4. *Let n_0 be the number of processes inside the system at time t_0. Let $t \geq t_0$, let $\Delta t = [t, t+\Delta]$ a time interval and let $\sum_{\tau=t+\Delta-3\delta+1}^{t+\Delta} \mu(\tau) < N(t+\Delta-3\delta)\ \forall t$ then $|A(\Delta t)| > 0\ \forall t$.*

Proof. Considering that processes become active, in the worse case, 3δ time after the invocation of their join and that in every time unit there is always at least one active process (due to Lemma 3), it follows that in every time interval there is always at least one active process. $\square_{Lemma\ 4}$

From this two Lemmas it is possible to derive a relation between the number of active processes and the refresh of the system.

Corollary 1. *Let $t \geq t_0$, let $\Delta t = [t, t+\Delta]$ a time interval.*

$$|A(\Delta t)| > 0 \Leftrightarrow L(\Delta t) < 1.$$

Proof From Lemma 3 and Lemma 4 we know that $|A(\Delta t)| \geq N(t+\Delta-3\delta) - \sum_{\tau=t+\Delta-3\delta+1}^{t+\Delta} \mu(\tau)$. In order to have at least one active process inside the system, we have to constraint such quantity to be greater than 0 and then

$$\begin{aligned} N(t+\Delta-3\delta) - \textstyle\sum_{\tau=t+\Delta-3\delta+1}^{t+\Delta} \mu(\tau) &> 0 \\ N(t+\Delta-3\delta) &> \textstyle\sum_{\tau=t+\Delta-3\delta+1}^{t+\Delta} \mu(\tau) \\ \frac{\sum_{\tau=t+\Delta-3\delta+1}^{t+\Delta} \mu(\tau)}{N(t+\Delta-3\delta)} &< 1, \end{aligned}$$

that is exactly the definition of $L(\Delta t)$. $\square_{Corollary\ 1}$

Now we are in the position to prove the safety of the implementation of the regular register. Informally, the following lemma states that to guarantee safety, during the join of a process, the number of processes departing from the system has to be strictly lesser than the ones joining the system.

Lemma 5. Safety. *Let the join function $\lambda(t)$, the leave function $\mu(t)$ such that $\sum_{\tau=t-3\delta+1}^{t} \mu(\tau) < N(t-3\delta)\ \forall t$ and consider the protocol described in Section 5. The read operation returns the last value written before the read invocation, or a value written by a write operation concurrent with it.*

Proof. The proof use the same structure as in [3] by replacing Lemma 1 of [3] with Lemma 3 and Lemma 4. $\square_{Lemma\ 5}$

Discussion. We are going now to show how the general model proposed can be applied to a specific case of churn and in particular we are going to show its application to the case of the churn allowing constant network size. Having a constant network size implies that the two thresholds k_1 and k_2 are equal to zero (i.e. no fluctuation from the initial value is allowed). Moreover, the join

distribution is the same as the leave distribution and in each time unit the same number of processes cn_0 enter and leave the system (i.e. $\lambda(t) = \mu(t) = cn_0$ for every t, where c is a percentage of nodes); hence, applying Definition 4 we obtain that $N(t) = n_0$ for every time t. If now we apply the churn constraint defined by Lemma 3 we have that

$$\begin{aligned} \textstyle\sum_{\tau=t-3\delta+1}^{t} \mu(\tau) &< N(t-3\delta) \\ 3\delta n_0 c &< n_0 \\ c &< \tfrac{1}{3\delta}, \end{aligned}$$

that is exactly the same constraint found in [3].

Another consideration that we can do is how to represent the infinite arrival model [1,17,20] using our model. In the infinite arrival model, the network size is at any time upper bounded by a known constant C and during the life of the system an infinite number of processes can be part of the computation. In our model, we can represent such a scenario simply saying that $k_2 = C - n_0$ and the join and the leave distribution are not always equal to zero. Consequence of our translation is that a regular register can be implemented in an infinite arrival model bounded by C if and only if the join and the leave distributions satisfy the constraints of Lemma 4.

7 Related Work

Dynamicity Model. Dynamic systems are nowadays an open field of research and then new models able to capture all the aspects of such dynamicity are going to be defined. In [1,17] are presented models, namely *infinite arrival models*, able to capture the evolution of the network removing the constraint of having a predefined and constant size n. However such models do not give any indication on how the joins or the leaves happen during time. More recently, other models have been proposed that take into account the process behavior. This is done by considering both probabilistic distribution [13], or deterministic distribution [9], on the join and leave of nodes (but in both cases the value of the system size is constant).

Churn in P2P Networks. The study of the churn phenomenon received many attention in the last years, especially in the context of peer-to-peer (p2p) networks with respect to the connectivity maintenance problem. The general approach to maintain connectivity in p2p systems is by using overlay management protocols (OMPs), that is protocols that arrange the participant peers in an overlay that can be both structured or unstructured . In [10,11] two examples of structured overlays are proposed by using a dynamic distributed hash table where peers may join and leave at any time. The authors consider the worse case scenario in which join and leave operations may happen and design a (synchronous) system having complete visibility and able to maintain desirable properties such as a low peer degree and a low network diameter. In particular a bound of $O(logn)$ worst-case joins and/or crashes per constant time interval is provided.

We follow a similar approach, computing the bound on the churn that make the implementation of the regular register correct and abstract the effects of the churn at the overlay level by assuming that the network remains connected and setting opportunely the value of the maximum transmission delay δ.

Regular Register Implementation in Dynamic Environment. To the best of our knowledge the proposed regular register protocol is the first for distributed systems subject to churn (as defined in Section 3). Other register protocols have been designed for mobile ad-hoc networks, e.g., [4,18,21]. Interestingly these protocols rely on a a form of deterministic broadcast, namely, geocast which is similar to our broadcast primitive in the sense that it ensures delivery to the processes that stay long enough in the system without leaving.

Another interesting approach is the one used in [15] where an atomic read/write object is implemented in a dynamic asynchronous network prone to failures. In [15], the dynamicity is managed by replicating the objects and atomicity is ensured by using quorums. In our approach, every process maintains a copy of the object (which is not replicated) and the dynamicity is managed by a specific join procedure that allows to "transfer" the current state of the object to joining processes. Moreover, in [15] a quorum of processes is needed to maintain the consistency in spite of changes while in our approach we have not such an assumption; in fact we do not assume a number of processes that have to remain in the system during changes but we find which are the bounds on the join and leave distributions that make the implementation correct.

8 Conclusion

In modern distributed systems the notion of processes continuously departing and joining the system (churn) is actually part of the system model and creates additional unpredictability to be mastered by a distributed application. Hence, there is the need to capture the churn of a dynamic system through tractable realistic models in order to prove formally the correctness of distributed applications running in such environments. Churn models used till now are mainly probabilistic based either on traces (e.g. [8]) or on node distribution (e.g. [7]). This paper has presented a generic model for a churn notion based on deterministic joining and departure distributions. This model is general enough to have the infinite arrival model and the constant size system as special cases. Based on an implementation of a regular register presented in [3], the paper proved the additional constraints that have to be satisfied for an implementation to be correct despite the fact that the churn varies in a given interval.

Acknowledgments

The authors want to thanks Matthieu Roy and Francois Bonnet for the insightful discussion had on registers in mobile networks. This work is partially supported by the European STREP projects COMIFIN and SM4All and by the European Network of Excellence ReSIST.

References

1. Aguilera, M.K.: A Pleasant Stroll Through the Land of Infinitely Many Creatures. ACM SIGACT News, Distributed Computing Column 35(2), 36–59 (2004)
2. Baldoni, R., Bertier, M., Raynal, M., Tucci, S.: Looking for a Definition of Dynamic Distributed Systems. In: Malyshkin, V.E. (ed.) PaCT 2007. LNCS, vol. 4671, pp. 1–14. Springer, Heidelberg (2007)
3. Baldoni, R., Bonomi, S., Kermarrec, A.M., Raynal, M.: Implementing a Register in a Dynamic Distributed System. In: To appear in Proc. 29th IEEE Int'l Conference on Distributed Computing Systems (ICDCS 2009), June 2009. IEEE Computer Society Press, Montreal (2009), ftp://ftp.irisa.fr/techreports/2008/PI-1913.pdf
4. Dolev, S., Gilbert, S., Lynch, N., Shvartsman, A., Welch, J.: Geoquorum: Implementing Atomic Memory in Ad hoc Networks. In: Fich, F.E. (ed.) DISC 2003. LNCS, vol. 2848, pp. 306–320. Springer, Heidelberg (2003)
5. Friedman, R., Raynal, M., Travers, C.: Abstractions for Implementing Atomic Objects in Distributed Systems. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 73–87. Springer, Heidelberg (2006)
6. Hadzilacos, V., Toueg, S.: Reliable Broadcast and Related Problems. In: Distributed Systems, pp. 97–145. ACM Press, New York (1993)
7. Godfrey, B., Shenker, S., Stoica, I.: Minimizing churn in distributed systems. In: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), pp. 147–158 (2006)
8. Gummadi, P., Dunn, R., Saroiu, S., Gribble, S., Levy, H., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 314–329 (2003)
9. Ko, S., Hoque, I., Gupta, I.: Using Tractable and Realistic Churn Models to Analyze Quiescence Behavior of Distributed Protocols. In: Proc. 27th IEEE Int'l Symposium on Reliable Distributed Systems, SRDS 2008 (2008)
10. Kuhn, F., Schmid, S., Wattenhofer, R.: A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In: Castro, M., van Renesse, R. (eds.) IPTPS 2005. LNCS, vol. 3640, pp. 13–23. Springer, Heidelberg (2005)
11. Kuhn, F., Schmid, S., Smit, J., Wattenhofer, R.: A Blueprint for Constructing Peer-to-Peer Systems Robust to Dynamic Worst-Case Joins and Leaves. In: Proceeding of 14th IEEE International Workshop on Quality of Service, IWQoS (2006)
12. Lamport, L.: On Interprocess Communication, Part 1: Models, Part 2: Algorirhms. Distributed Computing 1(2), 77–101 (1986)
13. Leonard, D., Yao, Z., Rai, V., Loguinov, D.: On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. IEEE/ACM Transaction on Networking 15(3), 644–656 (2007)
14. Liben-Nowell, D., Balakrishnan, H., Karger, D.R.: Analysis of the Evolution of Peer-to-peer Systems. In: 21th ACM Symp. PODC, pp. 233–242. ACM Press, New York (2002)
15. Lynch, N., Shvartsman, A.: RAMBO: A Reconfigurable Atomic Memory Service for Dynamic Networks. In: Malkhi, D. (ed.) DISC 2002. LNCS, vol. 2508, pp. 173–190. Springer, Heidelberg (2002)
16. Mostefaoui, A., Raynal, M., Travers, C., Peterson, S.: From Static Distributed Systems to Dynamic Systems. In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS 2005), pp. 109–119. IEEE Computer Society Press, Los Alamitos (2005)

17. Merritt, M., Taubenfeld, G.: Computing with Infinitely Many Processes. In: Herlihy, M.P. (ed.) DISC 2000. LNCS, vol. 1914, pp. 164–178. Springer, Heidelberg (2000)
18. Roy, M., Bonnet, F., Querzoni, L., Bonomi, S., Killijian, M.O., Powell, D.: Geo-Registers: an Abstraction for Spatial-Based Distributed Computing. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 534–537. Springer, Heidelberg (2008)
19. Tucci Piergiovanni, S., Baldoni, R.: Connectivity in Eventually Quiescent Dynamic Distributed Systems. In: Bondavalli, A., Brasileiro, F., Rajsbaum, S. (eds.) LADC 2007. LNCS, vol. 4746, pp. 38–56. Springer, Heidelberg (2007)
20. Tucci-Piergiovanni, S., Baldoni, R.: Eventual Leader Election in the Infinite Arrival Message-passing System Model. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 518–519. Springer, Heidelberg (2008), http://www.dis.uniroma1.it/~midlab/articoli/DISC08-tucci.pdf
21. Tulone, D.: Ensuring strong data guarantees in highly mobile ad hoc networks via quorum systems. Ad Hoc Networks 5(8), 1251–1271 (2007)

Ordered Coloring Grids and Related Graphs

Amotz Bar-Noy[1,2], Panagiotis Cheilaris[3,⋆], Michael Lampis[2], Valia Mitsou[2], and Stathis Zachos[1,2,4]

[1] Computer and Information Science Department
Brooklyn College
City University of New York
2900 Bedford Avenue
Brooklyn NY 11210 USA
amotz@sci.brooklyn.cuny.edu
[2] Doctoral Program in Computer Science
The Graduate Center
City University of New York
365 5th Avenue
New York NY 10016 USA
mlampis@gc.cuny.edu, vmitsou@cs.gc.cuny.edu
[3] Alfréd Rényi Institute of Mathematics
Hungarian Academy of Sciences
13–15 Reáltonoda utca
Budapest H-1053 Hungary
philaris@renyi.hu
[4] School of Electrical and Computer Engineering
National Technical University of Athens
Iroon Polytechneiou 9
Athens 15780 Greece
zachos@cs.ntua.gr

Abstract. We investigate a coloring problem, called ordered coloring, in grids and some other families of grid-like graphs. Ordered coloring (also known as vertex ranking) is related to conflict-free coloring and other traditional coloring problems. Such coloring problems can model (among others) efficient frequency assignments in cellular networks. Our main technical results improve upper and lower bounds for the ordered chromatic number of grids and related graphs. To the best of our knowledge, this is the first attempt to calculate exactly the ordered chromatic number of these graph families.

Keywords: grid graph, ordered coloring, vertex ranking, conflict-free coloring.

⋆ Part of the research was completed while the author was at the Graduate Center of the City University of New York.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 30–43, 2010.

1 Introduction

In this paper we focus on the problem of computing efficient *ordered colorings* (also known as *vertex rankings*) for grids and related graphs. Ordered colorings are defined as follows:

Definition 1. *An* ordered coloring *of* $G = (V, E)$ *with* k *colors is a function* $C\colon V \to \{1, \ldots, k\}$ *such that for each simple path* p *in* G *the maximum color assigned to vertices of* p *occurs in* exactly one *vertex of* p.

The problem of computing ordered colorings is a well-known and widely studied problem (see e.g. [9]) with many applications including VLSI design [10] and parallel Cholesky factorization of matrices [11]. The problem is also interesting for the Operations Research community, because it has applications in planning efficient assembly of products in manufacturing systems [8]. In general, it seems the vertex ranking problem can model situations where interrelated tasks have to be accomplished fast in parallel (assembly from parts, parallel query optimization in databases, etc.)

Another motivation for the study of ordered colorings comes from more recent research into an area of coloring problems inspired by wireless mobile networks, called conflict-free colorings. The study of conflict-free colorings originated in the work of Even et al. [6] and Smorodinsky [14]. Conflict-free coloring models frequency assignment for cellular networks. A cellular network consists of two kinds of nodes: *base stations* and *mobile agents*. Base stations have fixed positions and provide the backbone of the network; they are modeled by vertices in V. Mobile agents are the clients of the network and they are served by base stations. This is done as follows: Every base station has a fixed frequency; this is modeled by the coloring C, i.e., colors represent frequencies. If an agent wants to establish a link with a base station it has to tune itself to this base station's frequency. Since agents are mobile, they can be in the range of many different base stations. To avoid interference, the system must assign frequencies to base stations in the following way: For any range, there must be a base station in the range with a frequency that is not reused by some other base station in the range. One can solve the problem by assigning n different frequencies to the n base stations. However, using many frequencies is expensive, and therefore, a scheme that reuses frequencies, where possible, is preferable. Conflict-free coloring problems have been the subject of many recent papers due to their practical and theoretical interest (see e.g. [13,7,3,5,1]).

In the case where the ranges of the mobile agents are modeled by paths on the graph, the conflict-free coloring problem is very closely connected to the vertex ranking problem as defined above, since every path contains a uniquely colored vertex (i.e., a base station with a unique and maximum frequency). In fact, many approaches in the conflict-free coloring literature use unique maximum colorings (like ordered colorings) because the latter are easier to argue about. In addition, the topologies we study in this paper are of special interest in this setting because they can model frequency assignment in a Manhattan-like environment,where base stations are approximately placed on a regular grid

and this gives us additional motivation to calculate the exact ordered chromatic number of the grid.

In general graphs, finding the exact ordered chromatic number of a graph is NP-complete [12] and there is a $O(\log^2 n)$ polynomial time approximation algorithm [2], where n is the number of vertices. Since the problem is generally hard, it makes sense to study specific graph topologies and the focus of this paper is the calculation of the ordered coloring number of several grid-like families of graphs. Our main focus are grid graphs, which can be formally defined as follows:

Definition 2. *An* $m_1 \times m_2$ grid *is a graph with vertex set* $\{0, \ldots, m_1 - 1\} \times \{0, \ldots, m_2 - 1\}$ *and edge set* $\{\{(x_1, y_1), (x_2, y_2)\} \mid |x_1 - x_2| + |y_1 - y_2| \leq 1\}$.

In a standard drawing of the grid graph, vertex (x, y) is drawn at point (x, y) in the plane. The grid can also be defined as the *cartesian product* of two paths $P_{m_1} \times P_{m_2}$.

It is known [9] that for general planar graphs the ordered chromatic number is $O(\sqrt{n})$. Grid graphs are planar and therefore the $O(\sqrt{n})$ bound applies. One might expect that, since the graph families we study have a relatively simple and regular structure, it should be easy to calculate their ordered chromatic numbers. This is why it is rather striking that, even though it is not hard to show upper and lower bounds that are only a small constant multiplicative factor apart, the *exact* value of these ordered chromatic numbers is not known. The main contribution of this paper is to further improve on these upper and lower bounds and to the best of our knowledge this is the first such attempt.

Paper organization. In the rest of this section we provide the necessary definitions and some preliminary known results that will prove useful in the remainder. In Section 2 we present our results improving the known upper bounds on the ordered chromatic number of grids, tori and related graphs, while Section 3 deals with the lower bounds. Conclusions and open problems are presented in Section 4.

1.1 Preliminaries

First, let us remark that Definition 1 is not the typical definition found in the literature. Instead the more standard definition is:

Definition 3. *An* ordered k*-coloring of a graph* G *is a function* $C\colon V(G) \to \{1, \ldots, k\}$ *such that for every pair of distinct vertices* v, v', *and every path* p *from* v *to* v', *if* $C(v) = C(v')$, *there is an internal vertex* v'' *of* p *such that* $C(v) < C(v'')$. *The* ordered chromatic number *of a graph* G, *denoted by* $\chi_o(G)$, *is the minimum* k *for which* G *has an ordered* k*-coloring.*

It is not hard to show that the two definitions are equivalent (see for example [9]). We prefer to use Definition 1 because it is closer to the definition of conflict-free colorings. Conflict-free coloring can be seen as a relaxation of ordered coloring:

In every path there must be a uniquely colored vertex, but its color does not necessarily need to be the maximum occurring in the path.

A concept that will prove useful in the remainder (especially for proving lower bounds) is that of a graph minor.

Definition 4. *A graph X is a* minor *of Y, denoted as $X \preccurlyeq Y$, if there is a subgraph G of Y, and a sequence $G_0, \ldots, G_k$, with $G_0 = G$ and $G_k = X$, such that $G_i = G_{i-1}/e_{i-1}$, where $e_{i-1} \in E(G_{i-1})$ (i.e., edge e_{i-1} is* contracted *in G_{i-1}), for $i \in \{1, \ldots, k\}$. Edge contraction is the process of merging both endpoints of an edge into a new vertex, which is connected to all neighbors of the two endpoints.*

It is not difficult to prove, with the help of a recoloring argument, that the ordered chromatic number is monotone with respect to minors.

Proposition 1. *If $X \preccurlyeq Y$, then $\chi_o(X) \leq \chi_o(Y)$.*

In the rest of this section we provide ordered colorings for some graphs with (relatively) few edges.

Chain. Ordered coloring of a chain is equivalent to conflict-free coloring a chain and is better known as conflict-free coloring with respect to intervals [3]. Exactly, $1 + \lfloor \lg n \rfloor$ colors are needed: For $n = 2^k - 1$, the coloring is defined recursively as follows: The middle vertex receives the maximum color k so the left and right sides (with $2^{k-1} - 1$ vertices each) can freely use the same colors and are colored recursively.

Ring. To color a *ring*, we use the above coloring of a chain. We pick an arbitrary vertex v and color it with a unique and maximum color. The remaining vertices form a chain that we color with the method described above. This method colors a ring of n vertices with $2 + \lfloor \lg(n-1) \rfloor$ colors and it is not difficult to prove that this is optimal.

Grid. To color the $m \times m$ grid, denoted by G_m, we can use the previous idea of the recursive coloring. We simply divide the grid in 4 equal grids of half size and recursively color them using exactly the same colors for each. To make this possible we should use unique colors in the middle row and column, as we did for the middle vertex of the chain. So, we use m unique maximum colors for the middle row and then about $\frac{m}{2}$ unique colors for the middle column (the same above and under the middle row). This method requires about $3m$ colors. However, this coloring remains proper even if we add two edges in every internal face of the standard drawing of G_m. This indicates that $3m$ is not optimal and in fact, in section 2, we improve the above upper bound.

There is also a lower bound of $\chi_o(G_m) \geq m$ from [9]. Another proof (in [2]) is immediate from the fact that the *treewidth* and *pathwidth* of a graph G are at most the *minimum elimination tree height* (see [11] for the definition) of G. We provide yet another proof, based on minors:

Proposition 2. *If G_m is the $m \times m$ grid, $\chi_o(G_m) \geq m$.*

Proof. By induction. Base: For $m = 1$, it is true, as $\chi_o(K_1) = 1$. For the inductive step, consider a Hamilton path p of G_m, with $m > 1$. If G_m is ordered colored, then there is a vertex v with a unique color in p (and thus in G). So, for some v, $\chi_o(G_m) = 1 + \chi_o(G_m - v)$. However, for every v, $G_{m-1} \preccurlyeq G_m - v$. Therefore, from proposition 1, $\chi_o(G) \geq 1 + \chi_o(G_{m-1})$ and from the inductive hypothesis, $\chi_o(G) \geq 1 + m - 1 = m$.

In section 3, we improve the above lower bound.

2 Upper Bounds

In this and the next section we show how to color several grid-like families of graphs. We are mainly interested in the $m \times m$ (square) grid. In order to color the grid efficiently we rely on separators whose removal leaves some subgraphs of the grid to be colored. The subgraphs we will rely on are the rhombus R_x, the wide-side triangle T_x, and the right triangle O_x. These are depicted in Figures 1, 2, and 3 and formal definitions similar to definition 2 are not hard to infer. Another graph topology we will investigate is the *torus*, which is a variation of the grid with wraparound edges added, connecting the last vertex of every row (and column) with the first. The torus graph $\widehat{G}_m$ can also be defined as the cartesian product of two cycles $C_m \times C_m$. A summary of our upper bound results can be seen on Table 1. It is interesting that the golden ratio $\phi \approx 1.618$ appears in some of these bounds.

Table 1. Summary of upper bounds. The last column indicates on which upper bounds each result is based.

graph	upper bound	based on
G_m	$2.519\,m$	R_m, O_m
R_m	$1.500\,m$	-
T_m	$1.118\,m$	R_m
O_m	$1.618\,m$	T_m
$\widehat{G}_m$	$3.500\,m$	R_m

As was evident in the examples of the previous section, one strategy for constructing an ordered coloring of a graph is to attempt to find a separator, that is, a set of vertices whose removal disconnects the graph. The vertices of this set are all assigned distinct colors that will be the maximum colors used in the graph. This way, we can recursively construct a coloring for the components formed by the deletion of the separator, since paths connecting vertices from different components have a unique maximum vertex in the separator. The problem is then, to find a separator that is small and divides the graph into components of as low chromatic number as possible.

In the proofs we give below, we partition the graphs with the help of separators. All results are in the order of m, so without further mention we do not include terms logarithmic on m. These terms might be introduced by constant additive terms in a recursive bound. We are also omitting, in most cases floors and ceilings, because we are interested in asymptotic behavior. In that sense, a result like, for example, $\chi_o(G_m) \leq 2.67m$ should be read as an asymptotic upper bound of $2.67m \pm o(m)$.

In order to find improved upper bounds we need to find more intricate separators than those of the last example of the previous section. The idea is to use separators along diagonals in the grid. We will also need to find efficient colorings of some subgraphs that are left after we remove diagonal-like separators. That is the reason why we first present efficient colorings for the rhombus and the triangles.

In the figures of the following sections thicker lines indicate the selection of separator vertices which will receive unique and maximum colors. Thinner lines that lie on different sides of a thick line may reuse the same color range.

2.1 Rhombi and Triangles

The rhombus. The rhombus R_x is the first subgraph of the grid shown in Figure 1. It has height x. We have the following upper bound:

Proposition 3. $\chi_o(R_x) \leq 3x/2$.

Proof. Use a diagonal separator to cut the rhombus in half ($x/2$ unique colors are used), then cut also the remaining parts in half with a diagonal separator ($x/4$ unique colors, used in both parts). This is shown in figure 1. Therefore, we have the recursive formula $\chi_o(R_x) \leq x/2 + x/4 + \chi_o(R_{\lfloor x/2 \rfloor})$, which implies $\chi_o(R_x) \leq 3x/2$.

The wide side triangle. The triangle T_x is the subgraph of the grid shown in figure 2. Its long side has length x. First, we give a simple upper bound:

Proposition 4. $\chi_o(T_x) \leq 7x/6 \approx 1.167x$.

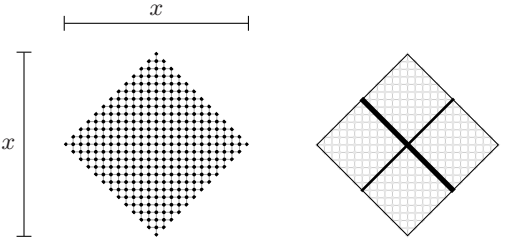


Fig. 1. The rhombus subgraph R_x and its separation

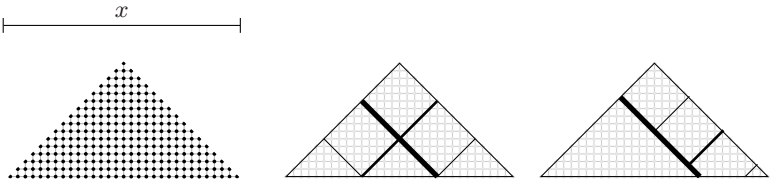


Fig. 2. The wide side triangle T_x and its separations

Proof. See the first separation of the wide-side triangle in Figure 2. Use a separator diagonally, parallel to one of the diagonal sides of the triangle T_x, with $2x/6$ unique colors. In the two remaining parts, separate diagonally by using separators parallel to the other diagonal side of the triangle T_x; each of those separators uses $x/6$ unique colors. With one more use of $x/6$ unique colors, we end up with connected components that are subgraphs of the rhombus $R_{2x/6}$. Therefore, $\chi_o(T_x) \leq 2x/6 + x/6 + x/6 + \chi_o(R_{\lfloor 2x/6 \rfloor})$, and since by proposition 3, $\chi_o(R_x) \leq 3x/2$, we have $\chi_o(T_x) \leq 7x/6$.

An improved upper bound can be obtained by the previous one, by making the observation that the graph on the left of the thickest separator in Figure 2 is also a wide side triangle. Thus, we may try to color it recursively in the same way. However, this would not improve the bound because the graph that remains on the right side uses $\frac{5x}{6}$ colors anyway. This indicates that the thickest separator would be better positioned if we moved it slightly to the right, since it seems that the remaining graph on the right side requires more colors.

Suppose that we move it slightly to the right, as in the last part of Figure 2 and that the ratio of its length over the length of the long side of the triangle is w (previously we had $w = 1/3$). We will optimize with respect to this w. Now, the rhombi on the right have length $x(1-2w)$, and the separators between them have length $x(1-2w)/2$. From the previously shown upper bound for the rhombus, and the fact that we need two sets of colors for the separators we conclude that the right part needs at most $\frac{5}{2}x(1-2w)$ colors. Assuming that the two parts are well balanced, the whole triangle needs at most $wx + \frac{5}{2}x(1-2w)$ colors. The triangle formed on the left of the separator has length $2wx$, thus from the above it needs $2w^2x + \frac{5}{2}(2wx)(1-2w)$ and in order for the balancing assumption to hold this must be equal to the number of colors used in the right part. Thus, we have $2w^2 + 5w(1-2w) = \frac{5}{2}(1-2w)$, which implies $w = \frac{5-\sqrt{5}}{8} \approx 0.345$. It is not hard to verify that using a separator of this length all the above arguments hold. Thus, we reach the following conclusion:

Proposition 5. $\chi_o(T_x) \leq \sqrt{5}x/2 \approx 1.118x$.

The right triangle. The right triangle O_x is the subgraph of the grid shown in figure 3. It has height x. We have the following upper bound:

Proposition 6. $\chi_o(O_x) \leq \phi x = \frac{\sqrt{5}+1}{2}x \approx 1.618x$.

Proof. See figure 3. Use a separator diagonally to form two wide side triangles whose long sides are of length x. We have the formula $\chi_o(O_x) \leq x/2 + \chi_o(T_x)$ and since by proposition 4, $\chi_o(T_x) \leq \sqrt{5}x/2$, we have $\chi_o(O_x) \leq \frac{\sqrt{5}+1}{2}x = \phi x$, where we denote by ϕ the golden ratio.

2.2 Grids and Tori

An $8m/3$ upper bound for square grids. In the first part of figure 4, we show how an $m \times m$ grid has to be partitioned with the help of separators to achieve an $8m/3$ upper bound.

The separators use m, $m/3$, and $m/3$ colors. After the removal of the separators, the remaining components are all subgraphs of a rhombus of height $2m/3$. By proposition 3, each remaining component can be colored with m colors. In total, $8m/3$ colors are required:

Proposition 7. $\chi_o(G_m) \leq 8m/3 \approx 2.6667m$.

An $18m/7$ upper bound for square grids. In the second part of figure 4, we show how an $m \times m$ grid has to be partitioned with the help of separators to achieve an $18m/7$ upper bound. The separators use m, $3m/7$, $3m/7$, $m/7$, and $m/7$ colors. Then, we have rhombi of height $2m/7$ that remain and, by proposition 3, each rhombus can be colored with $3m/7$ colors. In total, we have $18m/7$ colors:

Proposition 8. $\chi_o(G_m) \leq 18m/7 \approx 2.5714m$.

A $(7+2\phi)m/4$ upper bound for square grids. In the third part of figure 4 we show how an $m \times m$ grid can be partitioned to achieve a $(7 + 2\phi)m/4$ upper bound; we show only the partitioning of the subgraph under the first-level separator, since the subgraph over it is done in a symmetric way. We will show in the following section that shrinking this particular partition gives the best currently known result. The separators use $m + m/2 + m/4 = 7m/4$ unique colors. The remaining subgraphs of the grid to be colored are rhombi of height $m/2$ and right triangles of height $m/2$. By propositions 3 and 6 they can be colored with $3m/4$ and $\phi m/2$ colors respectively. Therefore the total use of colors is $7m/4 + \max(3m/4, \phi m/2) = (7 + 2\phi)m/4$.

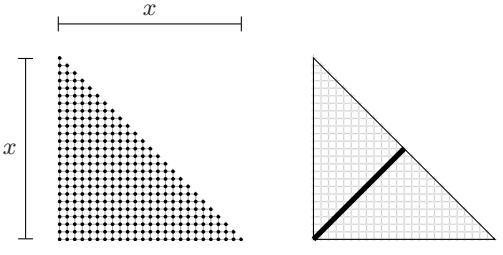


Fig. 3. The right triangle O_x and its separation

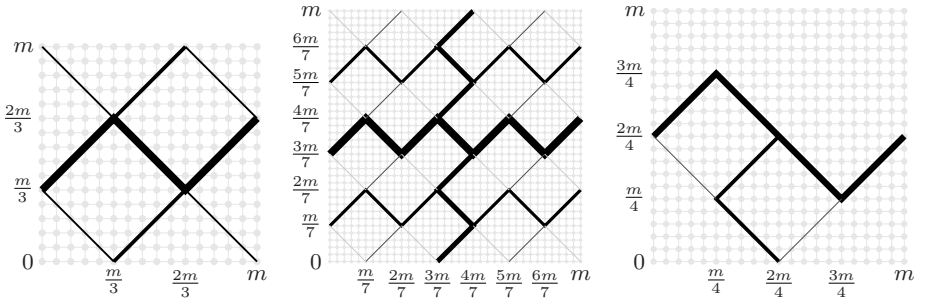


Fig. 4. $8m/3$, $18m/7$ and $(7+2\phi)m/4$ upper bounds

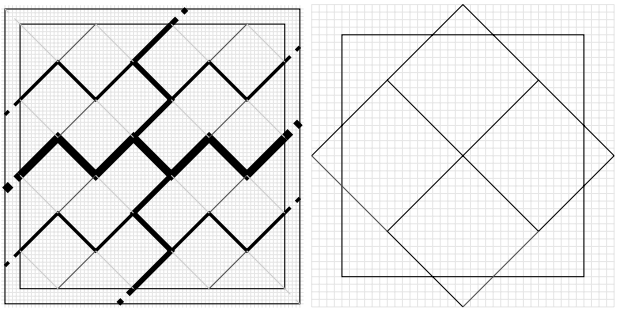


Fig. 5. A $18m/7$ coloring extended and a $(7+2\phi)m/4$ coloring shrunk

Improving the upper bounds by extending and shrinking colorings. The aforementioned upper bounds may be slightly improved by extending or shrinking the underlying grid. The reason is that, even though for the most part the grid is partitioned into rhombi, different subgraphs are formed at its edges.

In the case of the $8m/3$ and $18m/7$ bounds, we can see that wide side triangles are formed. For each of these we can use the same set of colors as for the rhombi formed further inside the grid, but since the rhombi are of twice the size of the triangles extending the grid to the point where the triangles use the same number of colors as the rhombi will not increase the total number of colors used. For example, for the $8m/3$ coloring, if the coloring is extended by $m\left(\frac{1}{\sqrt{5}}-\frac{1}{3}\right)$ in every side (up, down, left, right), then one can color the new grid of side length $m' \approx 1.228m$ with $\frac{4(13+3\sqrt{5})}{31}m'$ colors. Thus:

Proposition 9. $\chi_o(G_m) \leq \frac{4(13+3\sqrt{5})}{31}m \approx 2.544m.$

In the case of the $(7+2\phi)m/4$ coloring, we follow the opposite approach of shrinking the coloring. Four right triangles are formed, each using more colors than the rhombi. Therefore, slightly shrinking the grid so that the right triangles use the same number of colors as the rhombi improves the result. The optimal

amount of shrinking is $x = \left(\frac{1}{4} - \frac{3}{8\phi}\right) m$ from each side (up, down, left, right). The remaining grid has side $m' \approx 0.9635m$ and can be colored with $6\frac{\phi+1}{2\phi+3}m'$ colors. Thus, we get our best upper bound for the square grid:

Proposition 10. $\chi_o(G_m) \leq 6\frac{\phi+1}{2\phi+3}m \approx 2.519m$.

Torus. An efficient coloring of the torus $\widehat{G}_m$ is as follows: Use the two diagonals as separators (at most $2m$ vertices). The remaining two connected components are subgraphs of the rhombus R_m which can be colored with at most $3m/2$ colors. Therefore, we have the following proposition.

Proposition 11. $\chi_o(\widehat{G}_m) \leq 2m + 3m/2 = 3.5m$.

Rectangular grids. Intuitively, a rectangular grid begins to resemble a chain when one of its dimensions is much smaller than the other, i.e., $m_2 \gg m_1$. We may attempt to exploit this observation in the following manner: given a grid with m_1 rows and m_2 columns, pick the m_1-th column, the $2m_1$-th column, ..., the $(\lfloor m_2/m_1 \rfloor \cdot m_1)$-th column. These $\lfloor m_2/m_1 \rfloor$ columns will be used as separators, thus partitioning the graph into subgraphs of $m_1 \times m_1$ grids; each subgraph will use the same colors. However, the column separators do not all need distinct colors, because we can color them in a way similar to the coloring of a chain: the middle column receives the highest colors, then we color recursively the columns to the left and those to the right. This results to an upper bound of $\chi_o(G_{m_1,m_2}) \leq m_1 \lceil (1 + \log(\lfloor m_2/m_1 \rfloor)) \rceil + \chi_o(G_{m_1,m_1})$.

Moreover, the above upper bound can be further improved slightly. Instead of using columns as separators we may use a zig-zag line starting from the top left corner and proceeding diagonally to the right until it hits the bottom, the to the right and up again, and so on. This requires the same number of colors for the separators, since we can still color them in a chain-like fashion, but now wide-side triangles are formed (instead of grids), each of length $2m_1$, thus we reach the following conclusion:

Proposition 12. $\chi_o(G_{m_1,m_2}) \leq m_1 \left\lceil (1 + \log(\left\lfloor \frac{m_2}{m_1} \right\rfloor)) \right\rceil + \sqrt{5}m$

The above result is close to being optimal when $m_2 \gg m_1$ as we will see in the next section. However, when m_2 is not much larger than m_1, more careful strategies need to be examined.

3 Lower Bounds

In the first part of this section we prove lower bounds on the ordered chromatic number of square grids and tori. Then we move on to prove lower bounds for rectangular grids.

An important observation is the following: suppose we are given an optimal ordered coloring of a graph, and let $c_1, c_2, \ldots, c_k$ be the colors used, in decreasing

order. If c_i is the first color in this order assigned to more than one vertex, then vertices with colors $c_1, \ldots, c_{i-1}$ must form a separator, otherwise the path connecting the two vertices of color c_i would not have a unique maximum vertex. Thus, we can reason about a lower bound by reasoning about separators: examine cases on the size and shape of the separator formed by the highest colors of an optimal coloring and then, for each case, argue that the size of the separator plus the ordered chromatic number of one of the remaining components is higher than a desired lower bound. Moreover, it is enough to consider only *minimal separators*, as shown in [4] (a separator S is minimal if for every vertex v of S, $S \setminus \{v\}$ is not a separator).

In order to argue that the ordered chromatic number of a remaining component is high we will rely heavily on Proposition 1 and make use of induction.

We start with the torus lower bound, because the separators are simpler in this case.

Proposition 13. $\chi_o(\widehat{G}_m) \geq \frac{3m}{2}$ *(for* $m \geq 2$*).*

Proof. By induction: For $m = 2$ the proposition holds.

Suppose that we are given an optimal coloring of a torus $\widehat{G}_m$. Since the torus has no "sides" the separator must enclose an area of the torus. The smallest possible such separator is a set of the form $\{(x-1, y), (x, y+1), (x, y-1), (x+1, y)\}$, i.e., four vertices enclosing a single vertex (we call this kind of separator a cross). The length l of a separator will be $\max(|x_i - x_j| + 1)$ for $(x_i, y_i), (x_j, y_j)$ vertices of the separator. Similarly the height of a separator is $\max(|y_i - y_j| + 1)$. We distinguish between two cases:

Case 1: The separator formed by the highest colors encloses more than one vertex. Without loss of generality, suppose that the separator's length is at least as much as its height. Then the separator must consist of at least $2l - 2$ vertices. We also know that $l > 3 \Rightarrow l \geq 4$, otherwise the separator would enclose a single vertex only. Removing the separator will leave two components, one of which will have $\widehat{G}_{m-l}$ as a minor. Therefore, $\chi_o(\widehat{G}_m) \geq 2l-2+\chi_o(\widehat{G}_{m-l}) \geq \frac{3m}{2}+\frac{l}{2}-2 \geq \frac{3m}{2}$.

Case 2: The separator formed by the highest colors is a cross. It is not hard to see intuitively that this cannot lead to an optimal coloring, because our goal when using separators should probably be to balance the chromatic number of the components that will be formed, since only the maximum one matters. To show that this is the case, consider the following argument: let c_i be the color of vertex $(x, y-1)$, that is, a vertex outside the cross, but adjacent to two of its vertices. If it is unique, then $\chi_o(\widehat{G}_m) \geq 4 + 1 + \chi_o(\widehat{G}_{m-3})$, because the removal of the cross and this unique color leaves a graph with $\widehat{G}_{m-3}$ as a minor. Thus, $\chi_o(\widehat{G}_m) \geq 5 + \frac{3m}{2} - \frac{9}{2} > \frac{3m}{2}$. Now, if it is not unique it must be separated from its other appearances in the graph by a separator. If the separator is not a cross, similar reasoning as in case 1 proves the lower bound. If it is, we have two crosses contained in a 5×5 area. Therefore, $\chi_o(\widehat{G}_m) \geq 8 + \chi_o(\widehat{G}_{m-5}) > \frac{3m}{2}$.

We continue with a $4m/3$ lower bound for square grids, where the separators might also contain vertices on the sides of the grid (i.e., vertices with degree less than four).

Proposition 14. *For $m \geq 2$, $\chi_o(G_m) \geq \frac{4m}{3}$.*

Proof. Since we want to prove a $4m/3$ lower bound, we consider only separators of size $|S| \leq 4m/3$. The sides of the grid are the four paths of m vertices with $x = 0$, $x = m - 1$, $y = 0$, and $y = m - 1$, respectively. For the grid G_m we have the following cases of minimal separators.

Case I: The separator does not contain any vertex of the sides. This case is similar to the case of the torus. The size of the separator $|S| = s \geq 4$ and $G_m - S$ contains a $G_{m-(\lfloor s/2 \rfloor+1)}$ minor. Therefore, by induction, with such a separator, at least $s + (4/3)(m - (\lfloor s/2 \rfloor + 1)) \geq 4m/3$ colors are needed (because $s \geq 4$).

Case II: The separator touches at most two adjacent sides (i.e., sides that share a common vertex) of the grid. Then, $|S| = s \geq 2$ and $G - S$ contains a $G_{m-\lceil s/2 \rceil}$ subgraph. Therefore, by induction, with such a separator, at least $s + (4/3)(m - \lceil s/2 \rceil) \geq 4m/3$ colors are needed (because $s \geq 2$).

Case III: The separator touches two non-adjacent sides. In that case, the separator has size $|S| = s \geq m$. Consider the four square grid subgraphs $G_{\lceil m/2-s/6 \rceil}$ of the grid G_m that touch the four corners of G_m. It is not difficult to see that a separator of size s can not touch all four of the above subgraphs. Therefore, by induction, with such a separator, at least $s + \frac{4}{3}\lceil \frac{m}{2} - \frac{s}{6} \rceil \geq 4m/3$ colors are needed (because $s \geq m$).

Finally, we proceed to prove lower bounds for rectangular grids.

Proposition 15. $\chi_o(G_{m_1,2m_1}) \geq 2m_1$

Proof. By induction. For $m_1 = 1$ the proposition holds.

Let S be the separator formed by the highest colors. If $|S| \geq 2m_1$ then the proposition trivially holds. If $|S| < 2m_1$ then the separator can not touch both of the far sides of the grid. Thus, its removal will give us a component having height m_1. If $|S| < m_1$ the separator cannot touch two sides that are opposite each other. Therefore, its removal will give a graph with $G_{m_1-(|S|/2),2m_1-|S|}$ as a minor and thus $\chi_o(G_{m_1,m_2}) \geq |S| + 2m_1 - |S| = 2m_1$.

Finally, suppose that $m_1 \leq |S| < 2m_1$. The separator can not span a length of more than $|S|$ vertices, therefore one of the components formed must have $G_{(2m_1-|S|)/2,m_1}$ as a minor. Thus, $\chi_o(G_{m_1,2m_1}) \geq |S| + 2m_1 - |S| = 2m_1$.

Proposition 16. $\chi_o(G_{m_1,m_2}) \geq m_1 \left\lfloor \log\left(\frac{m_2}{m_1} + 1\right) \right\rfloor$

Proof. First, note that for $m_2 < 7m_1$ the proposition follows from previous propositions. Therefore, we will only deal with the case $m_2 \geq 7m_1$.

Let S be the separator formed by the highest colors. If $S < m_1$ then the removal of S must leave a component with $G_{m_1,m_2-|S|}$ as a minor. $\chi_o(G_{m_1,m_2}) \geq |S| + \chi_o(G_{m_1,m_2-|S|}) \geq |S| + m_1 \left\lfloor \log(\frac{m_2-|S|}{m_1} + 1) \right\rfloor > m_1 \left\lfloor \log(\frac{m_2}{m_1} + 1) \right\rfloor$.

If $m_1 \leq |S| \leq m_2 - 2m_1$ then, as in the previous proof, at least one $G_{m_1,(m_2-|S|)/2}$ minor is formed. Thus, $\chi_o(G_{m_1,m_2}) \geq |S| + \chi_o(G_{m_1,(m_2-|S|)/2}) \geq$

$|S| + m_1 \left\lfloor \log(\frac{m_2-|S|}{2m_1} + 1) \right\rfloor$. It is not hard to verify, using elementary calculus, that the latter is minimized when $|S| = m_1$ in which case $\chi_o(G_{m_1,m_2}) \geq m_1 + m_1 \left\lfloor \log(\frac{m_2+m_1}{2m_1}) \right\rfloor = m_1 \left\lfloor \log(\frac{m_2}{m_1} + 1) \right\rfloor$.

Finally, for $m_2 - 2m_1 < |S|$, let $m_2 = km_1$. Then $|S| > (k-2)m_1$, while $\log(\frac{m_2}{m_1}+1) = \log(k+1)$. We have that, $\chi_o(G_{m_1,m_2}) \geq |S| > (k-2)m_1$, therefore if $k - 2 > \lfloor \log(k+1) \rfloor$ the proposition holds. But we know that $k \geq 7$, which satisfies the previous inequality.

4 Open Problems

The most important problem still left open is of course the exact value of $\chi_o(G_m)$. For small values of m the correct answer seems to be $2m - 1$, but maybe this is just an exception for small values of m, and asymptotics could be different and closer to $2.5m$. It would also be interesting to study lower bounds for the rhombus and the triangle subgraphs, and then combine them to improve the lower bound for the square grid.

Another area for future research may be the online version of the problem, where vertices of the grid are "activated" one by one, and the coloring must remain proper throughout the process. Relevant results in the case of chains can be found in [1,3].

References

1. Bar-Noy, A., Cheilaris, P., Smorodinsky, S.: Deterministic conflict-free coloring for intervals: from offline to online. ACM Transactions on Algorithms 4(4), 44:1–44:18 (2008)
2. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. Journal of Algorithms 18(2), 238–255 (1995)
3. Chen, K., Fiat, A., Kaplan, H., Levy, M., Matoušek, J., Mossel, E., Pach, J., Sharir, M., Smorodinsky, S., Wagner, U., Welzl, E.: Online conflict-free coloring for intervals. SIAM Journal on Computing 36(5), 1342–1359 (2007)
4. Deogun, J.S., Kloks, T., Kratsch, D., Müller, H.: On the vertex ranking problem for trapezoid, circular-arc and other graphs. Discrete Applied Mathematics 98, 39–63 (1999)
5. Elbassioni, K., Mustafa, N.H.: Conflict-free colorings of rectangles ranges. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 254–263. Springer, Heidelberg (2006)
6. Even, G., Lotker, Z., Ron, D., Smorodinsky, S.: Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. SIAM Journal on Computing 33, 94–136 (2003)
7. Har-Peled, S., Smorodinsky, S.: Conflict-free coloring of points and simple regions in the plane. Discrete and Computational Geometry 34, 47–70 (2005)
8. Iyer, A.V., Ratliff, H.R., Vijayan, G.: Optimal node ranking of trees. Information Processing Letters 28, 225–229 (1988)

9. Katchalski, M., McCuaig, W., Seager, S.: Ordered colourings. Discrete Mathematics 142, 141–154 (1995)
10. Leiserson, C.E.: Area-efficient graph layouts (for VLSI). In: Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 270–281 (1980)
11. Liu, J.W.H.: The role of elimination trees in sparse factorization. SIAM Journal on Matrix Analysis and Applications 11(1), 134–172 (1990)
12. Llewellyn, D.C., Tovey, C.A., Trick, M.A.: Local optimization on graphs. Discrete Applied Mathematics 23(2), 157–178 (1989)
13. Pach, J., Tóth, G.: Conflict free colorings. In: Discrete and Computational Geometry, The Goodman-Pollack Festschrift, pp. 665–671. Springer, Heidelberg (2003)
14. Smorodinsky, S.: Combinatorial Problems in Computational Geometry. PhD thesis, School of Computer Science, Tel-Aviv University (2003)

Sub-linear Universal Spatial Gossip Protocols*

Hervé Baumann[1] and Pierre Fraigniaud[1,2]

[1] University Paris Diderot
[2] CNRS
Herve.Baumann@liafa.jussieu.fr,
Pierre.Fraigniaud@liafa.jussieu.fr

Abstract. *Gossip* protocols are communication protocols in which, periodically, every node of a network exchanges information with some other node chosen according to some (randomized) strategy. These protocols have recently found various types of applications for the management of distributed systems. *Spatial* gossip protocols are gossip protocols that use the underlying spatial structure of the network, in particular for achieving the "closest-first" property. This latter property states that the closer a node is to the source of a message the more likely it is to receive this message within a prescribed amount of time. Spatial gossip protocols find many applications, including the propagation of alarms in sensor networks, and the location of resources in P2P networks. We design a sub-linear spatial gossip protocol for arbitrary graphs metric. More specifically, we prove that, for any graph metric with maximum degree Δ, for any source s and any ball centered at s with size b, new information is spread from s to all nodes in the ball within $O((\sqrt{b \log b} \log\log b + \Delta) \log b)$ rounds, with high probability. Moreover, when applied to general metrics with uniform density, the same protocol achieves a propagation time of $O(\log^2 b \log\log b)$ rounds.

Keywords: epidemic algorithm, information spreading, resource location.

1 Introduction

Gossip protocols are communication protocols in which, periodically, every node of a network exchanges information with some other node chosen according to some (randomized) strategy. These protocols are appealing for their simplicity and robustness, and have recently found several applications for various network and system tasks, such as, e.g., multicast [5,9], resource location [7,8], and distributed databases management [1,6]. In essence, a gossip protocol performs as follows. Let V be a (finite or infinite, but countable) set of nodes. The protocol is executed by all the nodes in parallel for the purpose of broadcasting information – hereafter called "gossip" – among all nodes in V. More precisely, nodes

* Both authors received additional supports from the ANR project "ALADDIN", from the INRIA project "GANG".

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 44–56, 2010.

execute infinitely the same actions, in rounds, and, at each round, every node $u \in V$ applies the following two instructions:

(1) select a node $v \in V$;
(2) send known gossips to v;

The communication between node u and the selected node v is achieved via some underlying point-to-point communication protocol which allows any pair of distinct nodes to communicate in V.

Gossip protocols differ according to (1) the way each node selects the recipient of its next point-to-point communication, and (2) the way each node chooses the gossips to be sent to that recipient. In this paper, we restrict our attention to the former point, in order to measure the impact of the node selection mechanism on the efficiency of gossip protocols. Thus, for the sake of simplicity, we assume that once the recipient v of a point-to-point communication has been selected by u, this latter node transmits to v all the gossips learnt so far. Although this assumption might be unrealistic in many contexts (because sending many gossips obviously creates congestion), it allows us to focus on the way information can spread solely as a function of the networking environment, and in absence of any hypotheses regarding the nature of the gossips. In fact, there exists several environments in which ignoring congestion created by simultaneous transmissions of many different gossips is realistic. This is typically the case of alarm spreading among nodes of a sensor network, in environments in which few nodes are expected to be simultaneously the sources of alarms (e.g., forest fires, car accidents, etc.).

Protocols that are oblivious to the past are usually preferred, for they are not sensitive to any events that occurred previously. In particular, if the protocol is oblivious, then a node recovering from a crash or a transient fault can restart the execution of the protocol from scratch, even if all local information were lost. Therefore, the selection of node v performed by an informed node u is preferably the result of a mechanism that is not depending on the past. Moreover, protocols that are also oblivious to the sources of the gossips are also preferred, for their simplicity and efficiency. In particular, by treating all sources the same, no information is required to be stored in the gossip packets, or at least the examination of this information is not required to decide to which node(s) each gossip must be forwarded.

One way to overcome the two above constraints (time obliviousness, and source independence) is to consider randomized algorithms. In fact, currently proposed practical gossip protocols [5] are based on randomized mechanisms, mostly because randomization also preserves the mechanism from possible changes in the environment.

The most popular gossip protocol is UNIFORM: at each round, every informed node u selects the recipient v uniformly at random among all nodes in V. This protocol is known to perform well in practice [5]. It has been formally analyzed by Frieze and Grimmett [4], and by Pittel [10]. In particular, the former authors have proved that a new gossip is, w.h.p., spread to all nodes in V in $O(\log n)$ rounds, where $n = |V|$. This completion time is asymptotically optimal because

the number of nodes aware of a given gossip can at most double at each round, and thus it takes at least $\Omega(\log n)$ rounds for all nodes to become aware of a new gossip. However, it was noticed by Kempe, Kleinberg, and Demers [7] that UNIFORM is not appropriate to contexts in which closest nodes to the source of a new gossip should preferably receive this gossip faster than nodes farther away. Such a requirement occurs typically in the context of resource location [11] in which users are aiming at finding the nearby copies of duplicated shared resources (e.g., movies). It also occurs in the aforementioned context of alarm spreading.

Kempe et al. [7] tackled the issue of designing gossip protocols satisfying that the closer a node is from a source, the more likely it is to receive a gossip from that source within a prescribed amount of time. In order to measure the distance sensitivity of a gossip protocol, they have considered its *propagation time* as a function of the distance to the source, in a metric space (V, δ). In such a metric space, we denote by V the (finite or infinite but countable) set of points, or nodes, and by δ the distance function between nodes. Kempe et al. have designed a gossip protocol, here called DENSITY, satisfying that, if the nodes in V are spread with uniform density in the D-dimensional Euclidean space $\mathbb{R}^D$ with $\mathcal{L}_k$ metric, then a new gossip is spread to nodes at distance d from any source s in $O(\log^{1+\epsilon} d)$ rounds, with probability at least $1 - \frac{1}{\log d}$. By uniform density, it is meant that there exist two positive constants β_1 and β_2 such that, for any $r \geq 1$, the number of nodes in any ball of radius r is at least $\beta_1 r^D$, and at most $\beta_2 r^D$.

Protocols whose performances are sensitive to the distances between the sources and the recipients, are called *spatial* gossip protocols. In order to compare the propagation times of different spatial gossip protocols, in possibly different metric spaces (V, δ), we must take into account the fact that the number of nodes at a given distance from a given node varies significantly from one metric space to another, and even within the same metric space. In a metric (V, δ), it is actually more convenient to define the propagation time as a function of the ranks of the nodes, where the rank of node u relative to another node s is the number of nodes whose distances from s are not larger than $\delta(s, u)$. Indeed, a gossip protocol cannot insure that a gossip reaches a node close to the source quickly if there is a huge number of other nodes that are even closer to that source.

So let us redefine the propagation time as a function of the node ranks. The ball of radius d centered at s is defined as

$$B(s, d) = \{u \in V, \delta(s, u) \leq d\}.$$

For any node s and any $b \geq 1$, let $T_s(b)$ be the random variable equal to the number of rounds it takes for a new gossip introduced at node s to reach all nodes in the smallest ball B centered at s satisfying $|B| \geq b$. We say that a gossip protocol has propagation time $f(b)$ for some function f if for any $s \in V$, $T_s(b) \leq O(f(b))$ with high probability[1].

[1] When we write "with high probability" here, we mean with probability at least $1 - O(1/b^\alpha)$ for some $\alpha > 0$, where α may appear in the constant of the expression $O(f(b))$.

Table 1. Time complexities of various gossip protocols: n denotes the number of nodes, b the ball size, Δ the maximum degree, and D the diameter

Protocol	Application	Propagation time	Completion time
UNIFORM	Arbitrary finite metric		$O(\log n)$ [4, 10]
DENSITY	Uniform density in $(\mathbb{R}^D, \mathcal{L}_k)$	$O(\log^{2+\epsilon} b)$ [7]	
LOCAL	Arbitrary graph metric	$O(b \log b)$	$O(\Delta(D + \log n))$ [2]
LOGSCALE	Arbitrary graph metric	$O((\sqrt{b \log b}. \log\log b + \Delta) \log b)$	
	Metric of uniform density	$O(\log^2 b \log\log b)$	

By definition of uniform density, the balls in the sub-metric $(V, \mathcal{L}_k)$ of $(\mathbb{R}^D, \mathcal{L}_k)$ induced by a set V of nodes spread out with uniform density in $\mathbb{R}^D$ have sizes polynomial in their radius. Using this fact, one can show that the gossip protocol in [7] has propagation time $O(\log^{2+\epsilon} b)$. This result yields the question of the existence of efficient gossip protocols (that is protocols with bounded propagation time) in arbitrary metrics, or at least in arbitrary graph metrics. Recall that a graph metric (V, δ) is determined by an undirected unweighted graph $G = (V, E)$, where the distance $\delta(u, v)$ between two nodes u and v is the length of a shortest path between u and v in G.

In graph metrics, a natural candidate for such a protocol is the one that uses only the links of the graph: each node selects the recipient of its next communication uniformly at random among its neighbors in the graph. We call this protocol LOCAL. This protocol has been analyzed in detail in [2], where it is shown that it completes in $O(\Delta(D + \log n))$ rounds, w.h.p., where Δ denotes the maximum degree of the nodes, and D denotes the diameter of the graph. If fact, it is not difficult to adapt results in [2] to show that the propagation time of LOCAL is $O(b \log b)$ (see Section A in the Appendix), hence proving the existence of a universal spatial gossip protocol. This bound is tight. Indeed, in the n-node star (an n-node tree with $n - 1$ leaves an one internal node called center), a gossip introduced at the center of the star will reach all nodes at distance 1 in time $\Omega(n \log n)$, by equivalence to the coupon collector problem.

The main objective of this paper is to design universal spatial gossip protocols with sub-linear propagation times.

Our Results

We design a universal gossip protocol, called LOGSCALE, and prove that, in graph metrics of maximum degree Δ, its propagation time is $O((\sqrt{b \log b}. \log\log b + \Delta) \log b)$ rounds. The performances of this protocol compared to the previously mentioned protocols are summarized in Table 1. LOGSCALE has a propagation time significantly smaller than LOCAL. In finite graph metrics, it has the same completion time (i.e., the time to inform all nodes) as LOCAL, following from the fact that, in expectation, LOGSCALE acts as LOCAL for half of the rounds. During the other half, every node selects the recipient of its transmission with a probability that scales with the logarithm of the ranks. In fact, by combining LOGSCALE with UNIFORM (every node acts as in one protocol with probability half, and as in the other protocol with probability half), we obtain a gossip

protocol with the same propagation time as LOGSCALE but with the same completion time as UNIFORM. Although designed for graph metrics, our protocol LOGSCALE can also be applied to arbitrary metric. In metrics of uniform density (i.e., the same framework as in [8]), LOGSCALE achieves the polylogarithmic propagation time $O(\log^2 b \log\log b)$ rounds.

The paper is organized as follows. The gossip protocol LOGSCALE is described in Section 2, and analyzed in Section 3. The performances of LOGSCALE in metrics of uniform density are presented in Section 4. Finally, Section 5 lists some concluding remarks. (Section A in the Appendix revisits Protocol LOCAL to prove that its propagation time $O(b \log b)$ rounds.)

2 The Gossip Protocol LOGSCALE

This section describes the protocol LOGSCALE. The only thing one needs to specify is the way a node u selects a node v at each round. This selection process is inspired from the augmentation process in [3], in the sense that it uses a set of balls of exponentially growing size in which nodes are selected. However, as opposed to [3], the parameter k determining the size 2^k of the considered ball is not chosen uniformly at random in $[1, \log n]$, but is chosen with a probability decreasing as $1/k$. Moreover, our selection process gives high weight to neighboring nodes, as opposed to [3] which tends to ignore those neighboring nodes.

At each round, with probability $1/2$, node v is selected uniformly at random among all the neighbors of u, and, with the remaining probability $1/2$, v is selected in one ball containing 2^k nodes, for some $k > 0$. More precisely, for $k \geq 1$, let $C_k(u)$ be a set of 2^k closest nodes from u. The set $C_k(u)$ is not uniquely defined because of nodes at equal distance from u, so here $C_k(u)$ denotes one of these sets of 2^k closest nodes, chosen arbitrarily. (Note that, in finite graph metrics, $|C_k(u)| = \min\{n, 2^k\}$). To select v, node u picks one $k \geq 1$, and then selects v uniformly at random in $C_k(u)$. The choice of k is however not uniform, and k is picked with probability

$$p_k = \frac{1}{\sigma} \frac{1}{k \cdot \log^2(1+k)}.$$

where σ is a constant normalizing factor (independent of any parameter) so that $\sum_{k\geq 1} p_k = 1$. Note that choosing a larger σ would allow us to deal with transmission failures. Nevertheless, for the sake of simplicity, we assume here that $\sum_{k\geq 1} p_k = 1$. Note also that $\int_1^{+\infty} \frac{dx}{x \log^2(1+x)}$ is finite: the role of the polylog factor is specifically to insure convergence. Replacing p_k by $\frac{1}{k^{1+\epsilon}}$ would also work, but would increase the propagation time.

To sum up, let us define, for any two nodes u and v, the parameter

$$r_u(v) = \min\{k \geq 1 \mid v \in C_k(u)\},$$

and let $\Pr[u \to v]$ denotes the probability that node u selects node v at a given round. Finally, let $\deg(u)$ denote the degree of node u, i.e., the number of its adjacent nodes (neighbors). Then the protocol works as follows.

Protocol LOGSCALE**:** Set

$$p_{u,v} = \frac{1}{\sigma} \sum_{k \geq r_u(v)} \frac{1}{2^k \cdot k \cdot \log^2(1+k)}.$$

and set

$$\Pr[u \to v] = \begin{cases} \frac{1}{2}\left(\frac{1}{\deg(u)} + p_{u,v}\right) & \text{if } u \text{ and } v \text{ are neighbors,} \\ \frac{1}{2}\, p_{u,v} & \text{otherwise} \end{cases}$$

3 Propagation Time of LOGSCALE

In this section, we prove our main result, namely:

Theorem 1. *For any graph metric (V, δ), and for any source node $s \in V$, protocol* LOGSCALE *satisfies that a message introduced at node s reaches all nodes in any ball of size b centered at s in less than $O((\sqrt{b \log b} \cdot \log\log b + \Delta) \log b)$ steps, with high probability.*

Proof. Let (V, δ) be a graph metric, let $s \in V$, and let B be a ball centered at s, containing $b = |B|$ nodes. We prove that, with high probability, all nodes in B receive a gossip from s, at most $O((\sqrt{b \log b}. \log\log b + \Delta) \log b)$ rounds after it appeared at s. Let $k = \lceil \log b \rceil$. We have

$$C_{k-1}(s) \subseteq B \subseteq C_k(s).$$

Let us fix $u \in C_k(s)$, and set

$$\nu = \lceil 2^{k/2} \sqrt{k} \log(1+k) \rceil.$$

For any node $x \in C_k(s)$, we define $D(x)$ as the set of the ν closest nodes from x in $C_k(s)$, where, in case of ties, node u enters $D(x)$ first. That is,

$$u \notin D(x) \Rightarrow \forall w \in D(x), \delta(s,w) < \delta(s,u).$$

Let $P = (s_0, s_1, \ldots, s_\ell)$ be a shortest path from $s_0 = s$ to $s_\ell = u$. Then let i be the smallest index such that $u \in D(s_i)$.

Claim. The expected number of rounds of LOGSCALE before s eventually selects a node $v \in D(s_i)$ is at most $2\sigma\nu$.

Proof

$$\begin{aligned} \Pr[s \to D(s_i)] &= \sum_{v \in D(s_i)} \Pr[s \to v] \\ &\geq \sum_{v \in D(s_i)} \frac{p_{s,v}}{2} \end{aligned}$$

$$\geq \frac{|D(s_i)|}{2\sigma} \sum_{j \geq k} \frac{1}{2^j j \log^2(1+j)}$$
$$\geq \frac{\nu}{2\sigma 2^k k \log^2(1+k)}$$
$$= \frac{1}{2\sigma\nu}.$$

Therefore, after an expected number of rounds $2\sigma\nu$, some node $v \in D(s_i)$ has received the gossip directly from s. This establishes the claim. ◇

We now bound the expected number of rounds for the gossip to reach u from the node $v \in D(s_i)$. Let us consider the two shortest paths $P(v, s_i) = (v_0, v_1, \ldots, v_r)$ from $v_0 = v$ to $v_r = s_i$, and $P(s_i, u) = (s_i, s_{i+1}, \ldots, s_\ell)$ from s_i to $u = s_\ell$. In order to analyze the expected propagation time of the message from v to u along $P(v, s_i)$ and $P(s_i, u)$, we observe that if $d = \delta(s_i, u)$ then

$$B(s_i, d-1) \subseteq D(s_i) \subseteq B(s_i, d+1). \tag{1}$$

The first inclusion follows from the fact that $u \in D(s_i)$, hence all nodes at distance less than $d = \delta(s_i, u)$ must be in $D(s_i)$ as well by definition of the sets $D(\cdot)$. To establish the second inclusion, we first note that $D(s_{i-1}) \subseteq B(s_{i-1}, d)$ because s_{i-1} is at distance $d+1$ from u, and $u \notin D(s_{i-1})$, which implies that no other node at distance $d+1$ from s_{i-1} can be in $D(s_{i-1})$. Now, $B(s_{i-1}, d) \subseteq B(s_i, d+1)$. Hence $D(s_{i-1}) \subseteq B(s_i, d+1)$, and thus $|B(s_i, d+1) \cap C_k(s)| \geq \lceil \nu \rceil$. Therefore $D(s_i) \subseteq B(s_i, d+1)$. Let us first concentrate on the propagation time along $P(v, s_i) = (v_0, v_1, \ldots, v_r)$.

Claim. The expected number of rounds of LOGSCALE to travel from node $v \in D(s_i)$ to s_i is at most $6(\Delta + \nu)$.

Proof. We use a fact observed in [2] stating that every node outside a shortest path in a graph can be adjacent to at most 3 nodes of the path. We apply this observation in our context as follows. Any node in $D(s_i)$ can be adjacent to at most 3 nodes of $P(v, s_i)$. The problem is that some nodes outside $D(s_i)$ may also be adjacent to nodes of $P(v, s_i)$. Nevertheless, by Equation 1, only nodes $v = v_0$, v_1, and v_2 may be adjacent to nodes outside $D(s_i)$. Indeed, for all $j \geq 3$, we have $v_j \in B(s_i, d-2)$ because $v \in D(s_i) \subseteq B(s_i, d+1)$. That is v_j cannot be at the frontier between $D(s_i)$ and $V \setminus D(s_i)$ for $j \geq 3$. Therefore,

$$\sum_{j=0}^{r-1} \deg(v_j) = \deg(v_0) + \deg(v_1) + \deg(v_2) + \sum_{j=3}^{r-1} \deg(v_j)$$
$$\leq 3\Delta + 3|D(s_i)|$$
$$\leq 3(\Delta + \nu).$$

Now, the degree of a node is equal to the expected number of rounds to travel one more step along the path. As a consequence, the expected number of rounds for the gossip to travel from v to s_i is at most twice that bound (because neighbors are selected with probability $\frac{1}{2}$), and thus at most $6(\Delta + \nu)$, as claimed. ◇

Let us now concentrate on the propagation time along the path $P(s_i, u) = (s_i, s_{i+1}, \ldots, s_\ell)$.

Claim. The expected number of rounds of LOGSCALE to travel from s_i to u is at most $2(3\nu + \Delta)$.

Proof. By Equation 1, all nodes s_j for $j = i, \ldots, \ell - 2$ cannot be at the frontier between $D(s_i)$ and $V \setminus D(s_i)$. As a consequence,

$$\begin{aligned}\sum_{j=i}^{\ell-1} \deg(s_j) &= \sum_{j=i}^{\ell-2} \deg(s_j) + \deg(s_{\ell-1}) \\ &\leq 3|D(s_i)| + \Delta \\ &\leq 3\nu + \Delta.\end{aligned}$$

As a consequence, the expected number of rounds for the gossip to travel from s_i to u is at most $2(3\nu + \Delta)$. ◇

Let $T_{s,u}$ be random variable counting the number of round for a gossip arising at s to reach u. From what precedes, we get that

$$\begin{aligned}\mathbb{E}T_{s,u} &\leq 2\sigma\nu + 6(\Delta + \nu) + 2(3\nu + \Delta) \\ &= (2\sigma + 12)\nu + 8\Delta.\end{aligned}$$

Now, let $\alpha > 1$. For $i = 1, \ldots, \alpha \log b$, let X_i be independent random variables identically distributed as $T_{s,u}$, and denoting the time taken by a gossip starting from s at round $2(i-1)\mathbb{E}T_{s,u}$ to reach u. Since, the decision taken at each node in LOGSCALE is oblivious from the past, independent from the message source, and independent from the decision taken at other nodes, we get that if there exists i such that $X_i \leq 2\mathbb{E}T_{s,u}$, then $T_{s,u} \leq (2\alpha \log b)\, \mathbb{E}T_{s,u}$. Therefore,

$$\Pr[T_{s,u} > 2\,\alpha\, \log(b)\, \mathbb{E}T_{s,u}] \leq \prod_{i=1}^{\alpha \log b} \Pr[X_i \leq 2\mathbb{E}T_{s,u}].$$

By Markov inequality, we get $\Pr[X_i > 2\mathbb{E}X_i] < 1/2$. Therefore,

$$\Pr[T_{s,u} > 2\,\alpha\, \log(b)\, \mathbb{E}T_{s,u}] < \frac{1}{2^{\alpha \log b}} \leq \frac{1}{b^\alpha}.$$

Thus, by union-bound

$$\Pr[\exists u \in B,\ T_{s,u} > 2\,\alpha\, \log(b)\, \mathbb{E}T_{s,u}] \leq |B| \frac{1}{b^\alpha} = \frac{1}{b^{\alpha-1}}.$$

Thus

$$\Pr[\forall u \in B,\ T_{s,u} \leq 2\,\alpha\, \log(b)\, \mathbb{E}T_{s,u}] \geq 1 - \frac{1}{b^{\alpha-1}}$$

which yields

$$\Pr[\forall u \in B,\ T_{s,u} \leq 2\,\alpha\, \log(b)\, ((2\sigma + 12)\nu + 8\Delta)] \geq 1 - \frac{1}{b^{\alpha-1}}$$

We complete the proof by noting that

$$\nu = \left\lceil \sqrt{2^k k \log^2(1+k)} \right\rceil \leq O(\sqrt{b \log b} \log\log b). \qquad \square$$

4 Application to Metrics with Uniform Density

Protocol LOGSCALE can also be applied to arbitrary metrics (not only graph metrics). For the protocol to run in arbitrary metrics, one simply modifies it by having the selection process defined by

$$\Pr[u \to v] = p_{u,v}$$

for any pair of nodes u, v. (There is no more condition on whether u and v are adjacent or not). We analyze LOGSCALE in the context of metrics of uniform density (cf. [7]). In this paper, we use the following definition.

Definition 1. *A metric (V, δ) has uniform density if there exists a constant c such that, for any $s \in V$, and any $k \geq 1$, we have: $\forall u, v \in V,\ u, v \in C_k(s) \Rightarrow v \in C_{k+c}(u)$.*

We shall prove that Protocol LOGSCALE performs faster in metrics with uniform density than the bound of Theorem 1. Before that, we note that Definition 1 generalizes the definition of uniform density defined in [7] for sub-metrics of $(\mathbb{R}^D, \mathcal{L}_k)$. Recall that this latter definition states that a metric (V, δ) consisting of points scattered in $\mathbb{R}^D$ has uniform density if there exist two positive constants β_1 and β_2 such that, for any $r \geq 1$, the number of nodes in any ball of radius r is at least $\beta_1 r^D$, and at most $\beta_2 r^D$.

Remark. If a sub-merics of $(\mathbb{R}^D, \mathcal{L}_k)$ has uniform density in the sense of the definition in [7], then it has uniform density in the sense of Definition 1.

Proof. Assume that there exist two positive constants β_1 and β_2 such that, for any $r \geq 1$, the number of nodes in any ball of radius r is at least $\beta_1 r^D$, and at most $\beta_2 r^D$. W.l.o.g., one can assume that $\beta_1 \leq 1$. We prove that, for

$$c = 2D + \left\lfloor \log(\frac{\beta_2}{\beta_1}) \right\rfloor$$

we have: $u, v \in C_k(s)$ implies $v \in C_{k+c}(u)$ for any u, v and s.

Consider u, v and s such that $u, v \in C_k(s)$. Let r_{min} be the smallest radius such that $C_k(s) \subseteq B(s, r_{min})$.

We first analyze the "general" case $r_{min} > 1$. Let $1 \leq r' < r_{min}$ be such that $r_{min} \leq r' + 1$. We have $|B(s, r')| < 2^k$ because $r' < r_{min}$. On the other hand, we have $\beta_1 r'^D \leq |B(s, r')|$ by uniform density (in the sense of [7]). Thus $r' \leq \left(\frac{2^k}{\beta_1}\right)^{1/D}$. Now, by the triangle inequality, $\delta(u, v) \leq 2r_{min}$. Thus we get:

$$\begin{aligned}|B(s,\delta(u,v))| &\leq |B(s,2r_{min})| \\ &\leq \beta_2\,(2r_{min})^D \\ &\leq \beta_2\,2^D(r'+1)^D \\ &\leq \beta_2\,2^D\left(\left(\frac{2^k}{\beta_1}\right)^{1/D}+1\right)^D \\ &\leq \beta_2\,2^{2D}\frac{2^k}{\beta_1}.\end{aligned}$$

Combining this latter inequality with the fact that $v \in B(s,\delta(u,v))$, we get that

$$v \in C_{\lfloor \log(\beta_2\ 2^{2D}\frac{2^k}{\beta_1})\rfloor}(u).$$

That is, $v \in C_{k+c}(u)$.

The "particular" case $r_{min} \leq 1$ can be treated similarly. We have

$$|B(s,\delta(u,v))| \leq |B(s,2r_{min})| \leq \beta_2\,2^D.$$

Thus $v \in C_{D+\lfloor \log(\beta_2)\rfloor}(u) \subseteq C_c(u) \subseteq C_{c+k}(u)$. □

Theorem 2. *For any metric (V,δ) with uniform density, and for any source node $s \in V$, protocol* LOGSCALE *satisfies that a message introduced at node s reaches all nodes in any ball of size b centered at s in less than $O(\log^2 b \log\log b)$ steps, with high probability.*

Proof. The proof follows the same guidelines as the analysis of protocol DENSITY in [7]. Let $s \in V$, and $t \in C_k(s)$ for some k such that $k - 2\log k \geq 2c$ where c is the constant appearing in Definition 1. Let

$$U_s = C_{\lceil k/2+\log k\rceil+c}(s) \text{ and } U_t = C_{\lfloor k/2+\log k\rfloor+c}(t).$$

Fix $\beta > 2$, and let $\mathcal{E}$ denotes the event "there exists at least one call from U_s to U_t occurring during at least one of βk consecutive steps".

Claim. We have $\Pr[\mathcal{E}] \geq 1 - \frac{1}{2^{\beta k}}$.

Proof. By the uniform density hypothesis, we have $C_k(s) \subseteq C_{k+c}(t)$. Therefore, $U_s \subseteq C_{k+c}(t)$. Therefore, again by the density hypothesis, we get that, for any $u \in U_s$, and any $v \in C_{k+c}(t)$, $v \in C_{k+2c}(u)$. Thus, in particular,

$$\forall u \in U_s,\ \forall v \in U_t,\ v \in C_{k+2c}(u).$$

Using that property, one can bound the probability p of a call from U_s to U_t. We have

$$1 - p = \prod_{u\in U_s, v\in U_t} (1 - p_{u,v}).$$

As in the proof of Theorem 1, one can easily check that $p_{u,v} \geq x$ where

$$x = \frac{1}{2\sigma 2^{k+2c}(k+2c)\log^2(1+k+2c)}.$$

Therefore

$$1 - p \leq (1 - x)^{|U_s|\,|U_t|} \leq e^{-x\,|U_s|\,|U_t|}$$

and thus

$$1 - p \leq e^{-k^2/(2\sigma(k+2c)\log^2(1+k+2c))}.$$

For k big enough, we get that $1 - p \leq 1/2$. As a consequence,

$$\Pr[\mathcal{E}] \geq 1 - \frac{1}{2^{\beta k}}$$

as claimed. ◇

Now, we prove the following claim:

Claim. For any $t \in C_k(s)$, t receives a message originated at s in time at most $\beta\, k\, g(k)$ with probability at least $1 - \frac{g(k)}{2^{\beta k}}$ where $g(k)$ is solution of the recurrence equation

$$g(k) = 1 + g(\lceil k/2 + \log k \rceil + c) + g(\lfloor k/2 + \log k \rfloor + 2c).$$

Proof. We establish the claim by induction. We consider three consecutive time intervals:

$$\begin{aligned} I_s &= [1\ ,\ \beta\, k\, g(\lceil k/2 + \log k \rceil + c)] \\ I &= [|I_s| + 1\ ,\ |I_s| + \beta\, k] \\ I_t &= [|I_s| + |I| + 1\ ,\ |I_s| + |I| + \beta\, k\, g(\lfloor k/2 + \log k \rfloor + 2c)]. \end{aligned}$$

By the bound we have previously derived on the event $\mathcal{E}$, we get that during the time interval I, there exists a node $u \in U_s$ which calls a node in $v \in U_t$ with probability at least $1 - \frac{1}{2^{\beta k}}$. By induction hypothesis, node u has received the message of source s during time interval I_s with probability $1 - \frac{g(\lceil k/2 + \log k \rceil + c)}{2^{\beta k}}$. Also, by induction hypothesis, as $t \in C_{\lfloor k/2 + \log k \rfloor + 2c}(v)$, node t has received the message of source v during time interval I_t with probability $1 - \frac{g(\lfloor k/2 + \log k \rfloor + 2c)}{2^{\beta k}}$. Therefore during a time interval of duration $|I_s| + |I| + |I_t| = \beta\, k\, g(k)$ node t has received a message of source s with probability

$$(1 - \frac{1}{2^{\beta k}})(1 - \frac{g(\lceil k/2 + \log k \rceil + c)}{2^{\beta k}})(1 - \frac{g(\lfloor k/2 + \log k \rfloor + 2c)}{2^{\beta k}})$$

and thus with probability at least $1 - \frac{g(k)}{2^{\beta k}}$. ◇

Now, it is easy to see that $g(k) \leq O(k \log k)$. Thus, applying the claim, we get that for any $t \in C_k(s)$, t receives a message originated at s in time at most $O(\beta k^2 \log k)$ with probability at least $1 - O(\frac{k \log k}{2^{\beta k}})$. The theorem follows by applying union bound on all t's in $C_k(s)$. □

5 Conclusion

In this paper, we have proved that there exists a universal gossip protocol for all graph metric, whose propagation time is $O((\sqrt{b \log b} \cdot \log \log b + \Delta) \log b)$. A natural question is whether this bound can be improved. In particular it would be quite informative to prove or disprove the existence of a universal gossip protocol with polylogarithmic propagation time $O(\log^{\alpha} b)$ for some $\alpha \geq 1$. Such a polylogarithmic propagation time can be achieved in specific metrics, namely those with uniform density. Another natural extension of this work would thus be to extend the result to arbitrary metric without any assumption on the density. In fact, even the existence of a gossip protocol with finite propagation time is not clear in this general context: UNIFORM has an unbounded propagation time, and DENSITY and LOGSCALE have polylogarithmic propagation times in metrics with uniform density, but their performances in arbitrary metrics are not known.

Acknowledgements

All the authors are thankful to George Giakkoupis for helpful discussions.

References

1. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. In: 6th ACM Symposium on Principles of Distributed Computing (PODC), pp. 1–12 (1987)
2. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. In: Asano, T., Imai, H., Ibaraki, T., Nishizeki, T. (eds.) SIGAL 1990. LNCS, vol. 450, pp. 128–137. Springer, Heidelberg (1990)
3. Fraigniaud, P., Gavoille, C., Kosowski, A., Lebhar, E., Lotker, Z.: Universal Augmentation Schemes for Network Navigability: Overcoming the $\sqrt{n}$-Barrier. In: 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 1–7 (2007)
4. Frieze, A., Grimmett, G.: The shortest-path problem for graphs with random arclengths. Discrete Applied Math. 10, 57–77 (1985)
5. Gupta, I., Kermarrec, A.-M., Ganesh, A.: Efficient Epidemic-Style Protocols for Reliable and Scalable Multicast. In: 21st Symposium on Reliable Distributed Systems (SRDS), pp. 180–189 (2002)
6. Kempe, D., Dobra, A., Gehrke, J.: Computing Aggregate Information using Gossip. In: 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS (2003)
7. Kempe, D., Kleinberg, J., Demers, A.: Spatial gossip and resource location protocols. In: 33rd ACM Symposium on Theory of Computing, pp. 163–172 (2001)
8. Kempe, D., Kleinberg, J.: Protocols and impossibility results for gossip-based communication mechanisms. In: Proc. 43rd IEEE Symp. on Foundations of Computer Science, pp. 471–480 (2002)
9. Luo, J., Eugster, P., Hubaux, J.-P.: Route driven gossip: probabilistic reliable multicast in ad hoc networks. In: 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 2229–2239 (2003)

10. Pittel, B.: On spreading a rumour. SIAM J. Applied Math. 47, 213–223 (1987)
11. Plaxton, G., Rajaraman, R., Richa, A.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In: 9th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 311–320 (1997)

Appendix

A Protocol LOCAL Revisited

In this section, we briefly revisit the protocol LOCAL analyzed in [2], and prove that its propagation time is $O(b \log b)$.

Proposition 1. *For any finite graph metric (V, δ), and for any source node $s \in V$, protocol* LOCAL *satisfies that a message introduced at node s reaches all nodes in any ball of size b centered at s in less than $O(b \log b)$ steps, with high probability.*

Proof. We use the same proof structure as in [2]. Let (V, δ) be a finite graph metric, let $s \in V$, and let B be a ball centered at s, containing $b = |B|$ nodes. We prove that, using LOCAL, all nodes in B receive a gossip from s at most $O(b \log b)$ rounds after it appeared at s, with high probability. For this, we use again the observation in [2] stating that every node outside a shortest path in a graph can be adjacent to at most 3 nodes of the path. Let $u \in B$, and $P = (u_0, u_1, \ldots, u_\ell)$ be a shortest path from s to u, with $u_0 = s$ and $u_\ell = u$. Any node in $B \setminus P$ can be adjacent to at most 3 nodes of P. A node outside B adjacent to P can only be adjacent to u_ℓ since otherwise it would be in B. Therefore

$$\sum_{i=0}^{\ell-1} \deg(u_i) \leq 3b.$$

From this bound, we get that if X_u denotes the random variable equal to the time it takes for a gossip to reach u, then $\mathbb{E}X_u \leq 3b$. Now, let $\alpha > 1$. For $i = 1, \ldots, \alpha \log b$, let Y_i be independent random variables identically distributed as X_u. Since, the decision taken at each node in LOCAL is oblivious from the past, independent from the message source, and independent from the decision taken at other nodes, we have

$$\Pr[X_u \geq 2\ \alpha\ \log(b)\ \mathbb{E}X_u] \leq \Pi_{i=1}^{\log b} \Pr[Y_i \geq 2\mathbb{E}X_u].$$

By Markov inequality, we get

$$\Pr[X_u \geq 2\ \alpha\ \log(b)\ \mathbb{E}X_u] \leq \left(\frac{1}{2}\right)^{\alpha \log b} = \frac{1}{b^\alpha}.$$

Thus $\Pr[X_u \geq 6\ \alpha\ b\ \log b] \leq 1/b^\alpha$. By union-bound, we get that

$$\Pr[\exists u \in B, X_u \geq 6\ \alpha\ b\ \log b] \leq \frac{1}{b^{\alpha-1}}$$

which completes the proof. □

Designing Hypergraph Layouts to GMPLS Routing Strategies*

Jean-Claude Bermond[1], David Coudert[1], Joanna Moulierac[1], Stéphane Pérennes[1], Ignasi Sau[1,2], and Fernando Solano Donado[3]

[1] Mascotte joint project , I3S(CNRS-UNS) INRIA, Sophia-Antipolis, France
[2] Applied Mathematics IV Department of UPC, Barcelona, Spain
[3] Institute of Telecommunications, Warsaw University of Technology, Poland

Abstract. All-Optical Label Switching (AOLS) is a new technology that performs packet forwarding without any Optical-Electrical-Optical (OEO) conversions. In this paper, we study the problem of routing a set of requests in AOLS networks using GMPLS technology, with the aim of minimizing the number of labels required to ensure the forwarding. We first formalize the problem by associating to each routing strategy a logical hypergraph whose hyperarcs are dipaths of the physical graph, called *tunnels* in GMPLS terminology. Such a hypergraph is called a *hypergraph layout*, to which we assign a cost function given by its physical length plus the total number of hops traveled by the traffic. Minimizing the cost of the design of an AOLS network can then be expressed as finding a minimum cost hypergraph layout.

We prove hardness results for the problem, namely for general directed networks we prove that it is NP-hard to find a $C \log n$-approximation, where C is a a positive constant and n is the number of nodes of the network. For symmetric directed networks, we prove that the problem is APX-hard. These hardness results hold even is the traffic instance is a partial broadcast. On the other hand, we provide an $\mathcal{O}(\log n)$-approximation algorithm to the problem for a general symmetric network. Finally, we focus on the case where the physical network is a path, providing a polynomial-time dynamic programming algorithm for a bounded number of sources, thus extending the algorithm given in [1] for a single source.

1 Introduction

All-Optical Label Switching (AOLS) [9] is an approach to route packets transparently and all-optically, thus allowing a speed-up of the forwarding. This very promising technology for the future Internet applications also brings new constraints and new problems. Indeed, since the forwarding functions are implemented directly at the optical domain, a specific correlator (device) is needed

* This work has been partly funded by the European project IST FET AEOLUS and the project "Optimization Models for NGI Core Network" (Polish Ministry of Science and Higher Education, grant N517 397334).

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 57–71, 2010.

for each optical label processed in the node. Therefore, it is of major importance to reduce the number of employed correlators in every node, implying a reduction in the number of labels (as referred in the rest of the paper) that are going to be used by the traffic. Due to its flexibility as a control plane and to the fact that it handles traffic forwarding, the Generic MultiProtocol Label Switching (GMPLS) is the most promising protocol to be applied in AOLS-driven networks.

In GMPLS, traffic is forwarded through logical connections called Label Switched Paths (LSPs). When GMPLS is used with packet-based network, packets are associated to LSPs by means of a label, or tag, placed on top of the header of the packet. In this way, routers - called Label Switched Routers (LSRs) - can distinguish and forward packets.

The GMPLS standards allow packets to carry a set of labels in their header, conforming a stack of labels. Even though a packet may contain more than one label, LSRs must only read the first (or top) label in the stack in order to take forwarding decisions. This helps to reduce both the number of labels that need to be maintained on the core LSRs and the complexity of managing data forwarding across the backbone.

Stacking labels and label processing, in general, are standardized by the following set of operations that an LSR can perform over a given stack of labels:

- SWAP: replace the label at the top by a new one,
- PUSH: replace the label at the top by a new one and then push one or more onto the stack, and
- POP: remove the label at top in the label stack.

The labels stored in the forwarding table are significant only locally at the node and swapped all along the LSP (see Fig. 1).

Solutions deployed by GMPLS for reducing the number of labels are *label merging* [4,11,13] (not discussed here) and *label stacking* [12,15]. With label stacking, when two or more LSPs follow the same set of links, they can be routed together "inside" a higher-level LSP, henceforth a *tunnel*. In order to setup a tunnel, multiple labels are placed in the packet's header.

Fig. 1 represents the general operations needed to configure a tunnel with the use of label stacking. At the entrance of the tunnel, λ PUSH are performed in order to route the λ units of traffic through the tunnel. Then, only one operation (either a SWAP or a POP at the end of the tunnel) is performed in all the nodes along the tunnel, regardless of λ. In this figure, a stack of size 2 is used to route the λ LSPs in one tunnel from node A to node E. The top label l is swapped and replaced at each hop: by l_1 at node B, by l_2 at node C, and is finally popped at node D. The λ units of traffic, at the exit of the tunnel at node E can end or follow different paths according to their bottom label k_i, for all $i \in \{1, 2, ..., w\}$ in the stack.

A consequence of the way in which the GMPLS operations can be configured at LSRs is the following: traffic can enter in any node of a tunnel but can exit in only one point, the last node of the tunnel. In other words, when some traffic is carried by a tunnel, it follows the tunnel until its end.

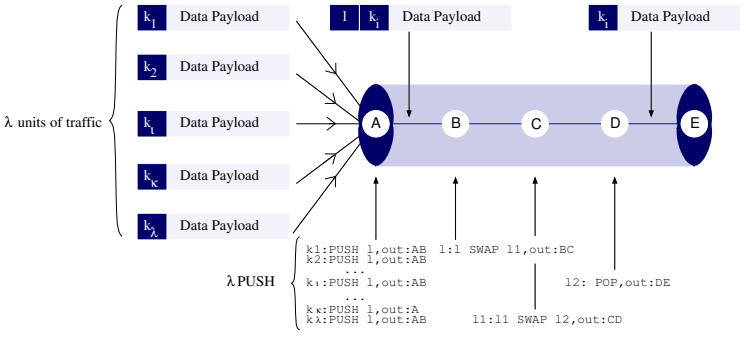


Fig. 1. GMPLS operations performed at the entrance and at the exit of a tunnel

Since the number of labels used for GMPLS forwarding affects the cost of the AOLS architecture, in this paper we mainly focus on the minimization of the number of labels used. In our previous example, the total cost $c(T)$ of this tunnel T from node A to node E in terms of number of labels is $c(T) = \lambda + \ell(T) - 1$, where λ is the number of units of traffic forwarded through this tunnel and $\ell(T)$ is its length in terms of number of hops (which is 4 on this example). We will formally define the cost function of the problem in Section 2.

Previous work and our contribution. The label minimization problem in GMPLS networks has been widely studied in the literature during the last few years [12,15,4,11,13,14]. All these articles focus mainly on proposing and analyzing heuristics to the problem, but there is a lack of theoretical results, like computational complexity or bounds on the approximation ratio of the proposed algorithms. For instance, in [14] the authors propose heuristics for routing a set of demands in AOLS networks when routers have limited number of available optical correlators. Very recently [1], the problem has been studied for the directed path from a more theoretical point of view. Namely, in [1] the authors present a polynomial-time optimal algorithm for the case when all traffic is issued from a single source and an $\mathcal{O}(\log n)$-approximation algorithm with arbitrary number of sources, where n is the number of nodes of the network.

In this article we provide the first theoretical framework for the label minimization problem in general GMPLS networks. We translate the problem into finding a set of dipaths in a directed hypergraph. With this new formulation, it turns out that the problem is very similar to classical Virtual Path Layout (VPL) problems originating from ATM networks. We provide hardness results and approximation algorithms for the problem in general graphs. The approximation algorithms strongly rely on the already known algorithms for VPL problems. Finally, we focus on the path topology, extending the dynamic programming approach presented in [1] to any bounded number of sources. If there are k sources, the main result is an optimal algorithm with running time $n^{\mathcal{O}(k)}$. That is, the problem is polynomial in the path for any fixed number of sources.

Organization of the paper. In Section 2 we formally state the problem in terms of hypergraph layout and fix the notation to be used throughout the article. In Section 3 we prove that for general directed networks it is NP-hard to find a $C \log n$-approximation, where C is a a positive constant and n is the number of nodes of the network. For symmetric directed networks, we prove that the problem is APX-hard, and therefore it does not accept a PTAS unless P=NP. In Section 4 we provide an approximation algorithm to the problem for symmetric directed graphs with an approximation ratio $\mathcal{O}(\log n)$, where n is the number of nodes of the network. In Section 5 we focus on the directed path topology and present a dynamic programming approach solving the problem in polynomial time when the number of sources is fixed. Finally, Section 6 is devoted to conclusions and further research.

2 GMPLS Logical Network Design as a Hypergraph Layout Problem

The logical network design problem that we address can be roughly described as follows: we are given a digraph (directed graph) G together with a set of traffic demands (or requests) between couples of vertices in G, and we must find a set of tunnels of minimum cost allowing to route all traffic requests. Note that usually communication networks are symmetric digraphs (i.e. when operators set a link on one direction, they also set the opposite link). So it is interesting to study the symmetric case, which turns out to be computationally easier than the general directed case. Let us now precise each one of the above terms.

A *tunnel* is simply a directed path (or dipath) in G, and due to the technological constraints discussed in Section 1, traffic can enter anywhere in the tunnel but must leave only at the end of the tunnel. To define the problem formally we need the following notation:

- $G = (V, E)$ is the underlying digraph (which can be symmetric or not).
- $|V| = n$, and vertices are numbered $1, \ldots, n$.
- r_{ij} is the request from $i \in V$ to $j \in V$, with multiplicity m_{ij}. R is the set of all requests.
- $P(G)$ is the set of all simple dipaths in G.
- t stands for a tunnel, and T is the set of tunnels, that is $t \in T \subseteq P(G)$.
- ℓ is a length function on the arcs, that is $\ell : E \to \mathbb{R}^+$.
- for a tunnel t, $\ell(t) = \sum_{e \in t} \ell(e)$ is its length and $w(t)$ is the amount of traffic it carries.

Note that a priori $w(t)$ depends on the routing policy. The cost of a tunnel t is then $w(t) + (\ell(t) - 1)$, and the cost of a set of tunnels T is

$$\sum_{t \in T} (w(t) + \ell(t) - 1)\,. \tag{1}$$

Each tunnel can be seen as a directed hyperarc on the vertex set of G. This observation naturally leads to the definition of a hypergraph layout.

Definition 1 (Hypergraph layout). *Given a graph G and a set $T \subseteq P(G)$, $H(T)$ is the directed hypergraph with $V(H(T)) = V(G)$, and where for each tunnel $t \in T \subseteq P(G)$ there is a directed hyperarc in $H(T)$ connecting any vertex of t to the end of t. $H(T)$ is called a* hypergraph layout.

Note that a hypergraph $H(T)$ defines a virtual topology on G. A hypergraph layout $H(T)$ is said to be *feasible* if for each request $r_{ij} \in R$ there exists a dipath in $H(T)$ from i to j. The problem can then be simply expressed as finding a feasible hypergraph layout of minimum cost. Let us now rewrite the cost function of Equation (1).

Given a hypergraph layout $H(T)$, let $L(r_{ij})$ be the number of hyperarcs that request r_{ij} uses, and let $d_H(i,j)$ be the distance from vertex i to vertex j in $H(T)$. Then the term $\sum_{t\in T} w(t)$ of Equation (1) can be rewritten as $\sum_{r_{ij}\in R} L(r_{ij})\cdot m_{ij}$ and, since $L(r_{ij}) \geq d_H(i,j)$, we conclude that in an optimal solution the routing necessarily uses shortest dipaths in the hypergraph layout. It follows that the cost function of Equation (1) can be rewritten w.l.o.g. as

$$\sum_{t\in T}(\ell(t)-1) + \sum_{r_{ij}\in R} d_H(i,j)m_{ij}. \tag{2}$$

The cost of a solution is of bicriteria nature. The first part is the cost of the hypergraph structure; we call it the *total length* of the layout. The second part is the total distance that the traffic travels in the hypergraph; we call it the *total hop count*. Both cost function parts are very much conflicting. On the one hand, to minimize the hop count, it is enough to take a shortest tunnel connecting any source to any destination. On the other hand, to minimize the total length of the layout, it is enough to use a minimum arc-weighted connected hypergraph H such that for each request $r_{ij} \in R$, vertices i and j lie on the same connected component of H. Summarizing, the problem can be stated as follows.

> Minimum Cost Hypergraph Layout: Given a digraph G with a length function and a set R of traffic requests, find a feasible hypergraph layout of minimum cost, where the cost of a hypergraph layout is defined as in Equation (2).

If G is a symmetric digraph, the problem is denoted Minimum Cost Symmetric Hypergraph Layout. It makes sense also to consider the decision version in which we are also given two positive integers C_L, C_H and the objective is to decide whether there exists a layout with total length less than C_L and total hop count less than C_H.

We note that the cost function of Equation (2) can be naturally generalized to

$$\alpha \cdot \sum_{t\in T} c(t) + \beta \cdot \sum_{r_{ij}\in R} d_H(i,j)m_{ij}\ , \tag{3}$$

where α and β are positive constants and $c(t)$ is a general cost function $c : P(G) \to \mathbb{R}^+$. The cost function of Equation (2) corresponds to $c(t) = \ell(t) - 1$.

Relation with VPL problems. This layout design problem defined above is quite similar to well studied VPL problems in ATM networks, in which one imposes a constraint on the logical structure and then wishes to minimize either the maximum distance [2] or the average distance [6] traveled by the traffic. Concerning hardness and approximation, we shall see in the sequel of the article that the problem we study inherits most of the characteristics of the classical VPL problems studied since the 80s. It is not surprising that, even if new technologies like GMPLS are proposed to cope with the increasing bandwidth of communication networks, the computational complexity of the problems associated to these technologies remains essentially the same.

Nevertheless, there are two crucial differences between the GMPLS problem that we study and the classical VPL version of ATM networks. Indeed, we have seen that the GMPLS logical network design problem can be translated into finding a set of dipaths in a *directed hypergraph*, whereas the existing models for VPL problems deal with *digraphs* without multiple arcs. This feature will be exploited in the dynamic programming approach for the path presented in Section 5. The second difference is that the cost function we consider takes into account the *sum* of the length and the hop count costs, whereas usually in VPL problems the aim is to minimize the *maximum* value of either the length or the hop count in the network. Finally, it is important to note that, if there is a single source in the the GMPLS version (or, more generally, if the traffic instance is such that in an optimal solution each hyperarc has exactly 2 vertices), then the problem is basically equivalent to a classical VPL problem.

3 Hardness Results

In this section we give hardness results for the Minimum Cost Hypergraph Layout problem. We distinguish two cases according to whether the underlying network is symmetric or not. We focus on those cases in Sections 3.1 and 3.2.

3.1 General Case

Theorem 1. *The* Minimum Cost Hypergraph Layout *problem cannot be approximated within a factor* $C \log n$ *for some constant* $C > 0$*, even if the instance is a partial broadcast, unless* P = NP.

Proof: The reduction is from Minimum Set Cover[1]. Raz and Safra [10] proved that Minimum Set Cover is not approximable within a factor $C \log n$, for some constant $C > 0$, unless P = NP. To a Set Cover instance with sets $S_1, S_2, \ldots, S_k$, with $S_i \subseteq \{a_1, a_2, \ldots, a_n\}$, we associate the following graph:

- We start with a distinguished node s.

[1] Given a finite set $\mathcal{S}$ and a collection $\mathcal{C}$ of subsets of $\mathcal{S}$, the aim is to find a subcollection $\mathcal{C}'$ of $\mathcal{C}$ of minimum cardinality that covers all the elements of $\mathcal{S}$.

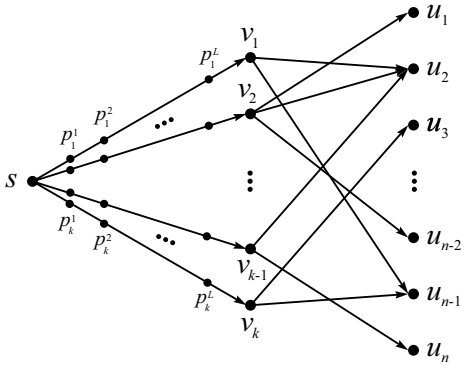


Fig. 2. Reduction in the proof of Theorem 1

- For each set S_i we introduce a node v_i and a directed path of length $L+1$ (L is a constant to be specified later) from s to v_i through L new vertices $p_i^1, p_i^2, \ldots, p_i^L$.
- For each element a_j we introduce a vertex u_j and, for each vertex v_i we add the arcs (v_i, u_j) if $a_j \in S_i$.
- The requests are from s to u_j, for $i = 1, \ldots, n$.

This construction is illustrated in Fig. 2. Let OPT be the optimal cost to the Minimum Cost Hypergraph Layout instance, and let OPT_{SC} be the optimal cost to the Minimum Set Cover instance.

Note that any cover defined by $I \subseteq \{1, 2, \ldots k\}$ induces a solution of Directed Hypergraph Layout obtained as follows: we use a tunnel of cost L connecting node s to each node $v_i, i \in I$ corresponding to a set taken in the cover. Then we connect each node $v_i, i \in I$ to the vertices $u_j, j \in S_i$. Finally, if a node u_j has more than one incoming tunnel (which means that a_j is covered more than once), we remove extra ones. A solution induced by an optimal cover has length cost $L \cdot OPT_{SC}$, and the hop count cost is $2n$, so $OPT \leq L \cdot OPT_{SC} + 2n$.

Conversely, given a layout, the dipaths from s to v_i used by some tunnel must induce a cover, so $OPT \geq L \cdot OPT_{SC} + n$. Putting all together,

$$L \cdot OPT_{SC} + n \leq OPT \leq L \cdot OPT_{SC} + 2n.$$

By choosing L to be large enough, the gap for the Minimum Cost Hypergraph Layout problem can be made as large as in Minimum Set Cover. Since, unless P = NP, approximating Minimum Set Cover within a factor $C \log n$ for some constant $C > 0$ is NP-hard [10], our result follows. □

3.2 Symmetric Case

When the input graph G is symmetric, we can consider G as an undirected where the edge $\{i, j\}$ corresponds to the two arcs (i, j) and (j, i).

Theorem 2. *The* Minimum Cost Symmetric Hypergraph Layout *problem is* APX*-hard even if the instance is a partial broadcast. Therefore, it does not accept a* PTAS *unless* P=NP.

Proof: The reduction is from Minimum Steiner Tree[2], which is know to be APX-hard [3], hence it does not accept a PTAS unless P = NP.

Given an instance $(G = (V, E), S \subseteq V)$ of Minimum Steiner Tree problem on n vertices, we build an instance of Minimum Cost Symmetric Hypergraph Layout problem by subdividing $\Omega\left(n^2 \cdot \sum_{r_{ij} \in R} m_{ij}\right)$ times each edge of G and considering as request set a partial broadcast from any vertex in S to all the others vertices in S. Note that subdividing edges is equivalent to setting $\alpha >> \beta$ in the cost function of Equation (3). In other words, the total hop count is negligible compared to the total length of the layout. It is then clear that any optimal solution to the Minimum Cost Symmetric Hypergraph Layout corresponds to a minimum cost Steiner tree in G spanning all the elements in S. Let OPT be the optimal cost to the Minimum Cost Symmetric Hypergraph Layout instance, and let OPT_{ST} be the optimal cost to the Minimum Steiner Tree instance. Let M be the number of times we have subdivided the edges of G. Summarizing,

$$OPT = M \cdot OPT_{ST} + o(M \cdot OPT_{ST}).$$

The existence of a PTAS for Minimum Cost Symmetric Hypergraph Layout would yield a PTAS for Minimum Steiner Tree, which is impossible unless P = NP. □

4 Approximation Algorithms

In this section we provide approximation algorithms for Minimum Cost Hypergraph Layout problem. Unless said otherwise, we focus on the symmetric version, for which the description of the algorithms is easier, although the main ideas could be adapted to the general version with slight modifications. For the sake of presentation, we describe our algorithms when the network is a path, a tree, and a general graph in Sections 4.1, 4.2, and 4.3, respectively. The approximation algorithm for the directed path network appeared also in [1], we include it here for the sake of completeness.

4.1 Case of the Path

First assume that the instance is a weighted all-to-all (i.e., there is a traffic request between each couple of nodes), and that n is a power of two (otherwise, just add dummy vertices). Then one simply uses the following binary layout: we connect

[2] Given an edge-weighted graph $G = (V, E)$ and a subset $S \subseteq V$, find a connected subgraph with minimum edge-weight containing all the vertices in S. We can assume, by subdividing edges, that all edge-weights equal 1.

node 1 to node $n/2$, node $n/2$ to node n, and we use recursively the binary layout for $n/2$ on the subdipaths $[1, n/2]$ and $[n/2, n]$. It is clear that any traffic request can be routed in this layout with at most $\log n$ hops, and that the total length of this layout is bounded above by $\log n \cdot \ell([1, n])$, where $\ell([1, n])$ denotes the length of the tunnel going from node 1 to node n. Therefore the cost of this layout is $\log n \cdot \sum_{r_{ij} \in R} m_{ij} + \log n \cdot \ell([1, n])$. Since any layout costs at least $\sum_{d \in D} m_{ij} + \ell([1, n])$, this provides a $\log n$-approximation in the all-to-all case.

Now, for a general traffic pattern, it is not always the case that $\ell([1, n])$ is a lower bound on the total length of the layout. We define the *span* of an instance as the minimum set of arcs such that any request can be routed using only those arcs. Note that the span is indeed a set of intervals such that any traffic request is routed within one of these intervals. Let ℓ_0 denote the length of the span. Then any layout costs at least $\sum_{r_{ij} \in R} m_{ij} + \ell_0$, and using the binary layout on each interval of the span we can define a layout with total length $\log n \cdot \ell_0$ and total hop count $\log n \cdot \sum_{r_{ij} \in R} m_{ij}$. Summarizing,

Proposition 1. *When the network is a path, there exists a polynomial-time approximation algorithm for* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

4.2 Case of the Tree

In [2] the authors studied the design of virtual layouts in ATM networks. Their model deals with point-to-point connections in the virtual graph, whereas in Minimum Cost Hypergraph Layout problem, a tunnel can carry more than one request. Nevertheless, we can use the results of [2] to obtain good approximation algorithms. Namely, we are interested in the following result which establishes the trade-off between the maximum load c and the diameter of a virtual layout allowing to route an all-to-all traffic in a general tree.

Theorem 1 (Bermond *et al.* [2]). *In a general tree on n nodes with all-to-all traffic, for each value of $c \in \{1, \ldots, n\}$ there exists a virtual layout allowing to route all traffic with diameter at most $10c \cdot n^{\frac{1}{2c-1}}$ and load at most c. In addition, such a layout can be constructed in polynomial time.*

In particular, if we set $c = \frac{\log n + 1}{2}$, Theorem 1 implies that we can find in polynomial time a layout with load $\mathcal{O}(\log n)$ and diameter at most $(5 \log n + 5) \cdot n^{\frac{1}{\log n}} = 10 \log n + 10 = \mathcal{O}(\log n)$.

Suppose first that the instance of Minimum Cost Hypergraph Layout problem is a weighted all-to-all traffic. It is clear that each arc must be used by some tunnel, hence $n - 1$ is a lower bound on the total length of any layout. On the other hand, the hop count is at least $\sum_{r_{ij} \in R} m_{ij}$. In the layout described above, each arc is used at most $\frac{\log n + 1}{2}$ times, and therefore the total length of this layout is $\mathcal{O}(n \log n)$. Since the diameter is also $\mathcal{O}(\log n)$, the total hop count is $\mathcal{O}(\log n \cdot \sum_{r_{ij} \in R} m_{ij})$, yielding an $\mathcal{O}(\log n)$-approximation.

If the instance is not all-to-all, we repeat the argument of the *span* discussed in Section 4.1, obtaining again an $\mathcal{O}(\log n)$-approximation. Summarizing,

Proposition 2. *When the network is a tree, there exists a polynomial-time approximation algorithm for* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

4.3 General Network

In the Minimum Generalized Steiner Network problem, we are given a graph $G = (V, E)$, a weight function $w : E \to \mathbb{N}$, a capacity function $c : E \to \mathbb{N}$, and a requirement function $r : V \times V \to \mathbb{N}$. The objective is to find a *Steiner network* over G that satisfies all the requirements and obeys all the capacities, i.e., a function $f : E \to \mathbb{N}$ such that, for each edge e, $f(e) \leq c(e)$ and, for any pair of nodes i and j, the number of edge disjoint paths between i and j is at least $r(i, j)$, where for each edge e, $f(e)$ copies of e are available. We want to minimize the cost of the network, i.e., $\sum_{e \in E} w(e) f(e)$. The problem is approximable within $\mathcal{O}(\log r_{\max})$, where $r_{\max}$ is the maximum requirement [7], and within a constant factor 2 when all the requirements are equal [8]. The directed version of the problem is approximable within $\mathcal{O}(n^{2/3} \log^{1/3} n)$ [5].

Given an instance of Minimum Cost Hypergraph Layout in a general network, consider the associated Minimum Generalized Steiner Network problem where all the requirements are equal to 1 and where the edge capacities are set to $+\infty$. Let H be an optimal solution to this Minimum Generalized Steiner Network instance (note that H may be disconnected). The following easy observation will be useful: since H is the smallest subgraph of G such that any couple source-destination lies on the same connected component, in any solution to the Minimum Cost Hypergraph Layout problem, the number of arcs that are used by at least one tunnel is at least $|E(H)|$. Using the algorithm of [8], we can find in polynomial time a Steiner network H' with $|E(H')| \leq 2|E(H)|$. Since the edge capacities are set to ∞, we can assume that such a Steiner network is a forest. The layout is then obtained by applying the algorithm described in Section 4.2 to each connected component of H'.

It is clear that the hop count of this layout is at most $\mathcal{O}(\log n)$ times the lower bound $\sum_{r_{ij} \in R} m_{ij}$. On the other hand, the total length of this layout is $\mathcal{O}(\log n \cdot |E(H')|) = \mathcal{O}(\log n \cdot |E(H)|)$. Since the total length of any layout is lower-bounded by $|E(H)|$, the $\mathcal{O}(\log n)$-approximation follows. Summarizing,

Theorem 2. *In a general network, there exists a polynomial-time approximation algorithm for* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

5 The Hypergraph Layout Problem on the Path

In this section we focus on the case when the underlying digraph is a directed path (nodes are numbered from left to right $1, \ldots, n$). Our approach consists in a dynamic programming algorithm that computes partial solutions induced on

subdipaths of the original path. We denote by $[i, j]$ the subpath from node i to node j.

Loosely speaking, we use the following dynamic program: we consider a cut vertex i and we look at a local solution induced on the subpath $[1, i]$. That is, the tunnels and traffic located on $[1, i]$. The cost of a local solution is defined as the sum of the local tunnels cost plus the hop counts sum taken on the local traffic.

We introduce then node $i+1$ and the potential tunnels finishing at it. In order to update the local solution cost, it is necessary to have enough information to compute the hop counts once this tunnel is introduced in the solution. So for each source $s \in S$ and vertex x, we introduce $h(s, x)$ defined as the hop count from s to x. Each vertex is then characterized by a hop count vector $h(x)$ whose dimension is the number of sources. A partial solution is then fully encoded by its local cost and the hop counts of all its nodes. It follows from the above discussion that we can encode a partial solution by giving, for each of its hop count vectors, the rightmost node associated to that vector. If we denote by h a bound on the hop count (at most n) and by c a bound (at most n) on the tunnel cost, we have $(c^k)^{h^k} = c^{kh^k}$ such possible table entries, where k is the number of sources.

By making an error of ε on the two costs (length and hops), we can encode the logarithm in base $1 + \varepsilon$ of those quantities, which leads to tables of size $\Theta\left((\log n)^{k(\log n)^k}\right)$. Note that this running time is already subexponential, so the problem is unlikely to be NP-hard to approximate within a constant factor when the number of sources is bounded (because it is widely assumed that algorithms solving 3-SAT require $2^{\Theta(n)}$ time). We shall see now how to improve this first naïve dynamic program.

We proceed now to give all the details for one and two sources, that suffice to get the intuition for an arbitrary number k of sources.

5.1 Case of a Single Source and the Non Crossing Property

We summarize the algorithm that appeared first in [1] (Gerstel *et al.* used a similar approach in [6]). In the case of a single source, it is not difficult to see that the tunnel structure is *non crossing*, i.e. two tunnels can only intersect in an optimal solution if one is strictly inside the other [1]. Since the path is directed, we assume w.l.o.g. that the source is located in the leftmost node of the path. This leads to the following approach: we consider the rightmost tunnel originating from the source and assume it ends at node i. Clearly, any tunnel starting in $[1, i-1]$ and ending in $[i, n]$ can be replaced with a tunnel starting at i, since this new tunnel may only decrease the hop count and the length[3].

This approach allows us to compute the optimal for a path with n vertices inductively. We denote by $C[i, j]$ the minimum cost for the requests destined to the subdipath $[i, j]$, in which the source is replaced by node i. Then for $2 \leq i \leq n$,

[3] This fact holds for any increasing cost function $c(t)$ in Equation (3).

$$C[1,i] = \min_{k<i} \left\{ C[1,k-1] + \left(\sum_{e \in E([1,k])} \ell(e) - 1 \right) + \sum_{j=k}^{i} m_{1j} + C[k,i] \right\}. \quad (4)$$

Note that $C[1,n]$ is the optimal cost of the original problem, and it can be computed in time $\mathcal{O}(n^3)$ [1]. In the particular case of the uniform broadcast (that is, $m_{ij} = 1$ for $i = 1$ and $2 \leq j \leq n$, and $m_{ij} = 0$ otherwise) and with a unitary length function on the edges, a closed formula was given in [1].

5.2 Case of Two Sources

We use a dynamic program similar to the one used for the single source case, but slightly more complicated. Let s_0 be the leftmost source and let s_1 be the other. In order to solve the problem, we introduce an auxiliary problem with *pseudo-sources*. A pseudo-source s is denoted by a triple (h_0, h_1, l), where l is the distance from s to the subdipath that lies on the right of the rightmost pseudo-source, and where h_i indicates that from s one can reach s_i in h_i hops, $i = 0, 1$. In the induction of the dynamic program the following auxiliary problem will appear:

- The traffic is restricted to an interval $[u, v]$, where either u or v is an end of the original dipath.
- There are one or two pseudo-sources located to the left of $[u, v]$.
- If there are two pseudo-sources, they are labeled (j, j, l_0) and $(j + 1, j, l_1)$, and we denote the corresponding problem $P((j, j), (j + 1, j), l_0, l_1, [u, v])$.
- If there is a single pseudo-source, it is labeled (j, k), and we denote the problem $P((j, k), [u, v])$.

In both cases we denote by OPT the cost of an optimal solution. Note that $P((0, 0), [u, v])$ is indeed a single source problem in which a unique source replaces both s_0 and s_1. Moreover, $P((j, k), [u, v])$ is equivalent to a single source problem, since $OPT(P((j, k), [u, v])) = OPT\left(P(0, 0, [u, v])) + \sum_{x \in [u,v]} (j \cdot m_{s_0 x} + k \cdot m_{s_1 x})\right)$. We now relate the two sources problem to the auxiliary problem. Consider the rightmost tunnel having s_i as head and denote by E_i its end node, $i = 0, 1$. We compute an optimal solution conditioned to the values E_i, $i = 0, 1$. There are three cases to consider, as it is depicted in Fig. 3.

(a) E_0 is left to s_1. Then on the subpath $[E_0, n]$ we pick an optimal solution with a slightly modified instance: we leave traffic requests toward s_1 unchanged and we replace the source s_0 by a pseudo-source at E_0 with hop count 1. On the subpath $[s_0, E_0 - 1]$ we use an optimal solution (note that in this subproblem there is only one source).

(b) E_0 is on the right of both s_1 and E_1. Then E_0 is at distance 1 from both sources and therefore any tunnel entering $[E_0 + 1, n]$ can be assumed to start at E_0. So the optimal solution is then obtained by using $OPT([s_0, E_0])$.

Note that in both cases (a) and (b) the induction is valid because E_0 is the best node to start a tunnel going to its right. Indeed, starting at E_0 is cheaper, and no node closer to the sources can be reached (from the definition of E_0).

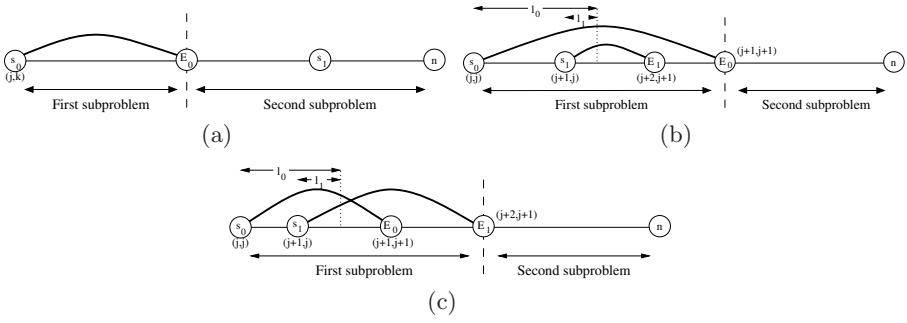


Fig. 3. Dynamic programming with two sources: cases (a), (b), and (c), respectively

(c) E_1 on the right of E_0. Note that E_0 is a $(1,1)$ pseudo-source, while E_1 is a $(2,1)$ pseudo-source. Consider a tunnel ending in $[E_1+1, n]$. The situation gets more complicated than in the single source case, since E_1 (a $(2,1)$ node) is not the "best" possible node anymore. The only nodes that can beat E_1 are $(1,1)$ nodes, and E_0 is the rightmost one. Hence we can assume that such a tunnel is starting either at E_0 or at E_1. Indeed, we have two "best nodes". So to perform the induction we have to solve two subproblems:

(c.1) the first subproblem on $[s_0, E_1-1]$, but under a condition on the location of the rightmost tunnel from s_1, i.e. $OPT([s_0, E_1-1] \mid (s_1, E_1))$.

(c.2) the second subproblem in which we have two pseudo-sources E_0 of type $(1,1)$ and E_1 of type $(2,1)$. So we pay $OPT(P((1,1),(2,1), l(E_0,E_1), l(E_1,E_1+1), [E_1+1,n]))$.

To complete our algorithm we need to show how to compute the dynamic program tables inductively, i.e. to compute $OPT(P((j,j),(j+1,j), l_0, l_1, [u,v]))$.

The two pseudo-sources tables. The induction is again on the two rightmost nodes E_0, E_1, with essentially the same cases as above, except case (a), which cannot occur since both pseudo-sources are now located outside the path.

(i) E_0 is on the right of E_1. Then E_0 is at distance $j+1$ from both sources and therefore any tunnel entering $[E_0+1, n]$ can be assumed to start at E_0. So the optimal solution is obtained by using $OPT([s_0, E_0])$ for the second subproblem and $OPT(P((j,j),(j+1,j), l_0, l_1, [s_0, E_0-1]))$.

(ii) E_0 is on the left of E_1. The situation is similar to case (c). We can split the problem into two subproblems, the first one being a two pseudo-sources problem reduced to $[u, E_1-1]$ with a condition on the rightmost tunnel from s_0, and the second being a single source problem on $[E_1+1, n]$.

Correctness & complexity. To complete the proof, we must explain how the above induction allows to compute all the tables inductively. Here are some explanations:

- First the induction is performed on the length of the path and when the tables for $[u,v]$ are computed, all the tables for strict subdipath of $[u,v]$ are known.

- Second, when filling the new tables, we compute the cost in a consistent way: we sum the cost of the first and second subproblems (found in already computed tables) with the cost of the tunnels that are removed, plus the hop count for traffic toward the removed node (either E_0 or E_1).
- As usual we keep only the best cost found when examining all the subcases 1,2,3.
- Finally, one may worry about the conditioning on the rightmost tunnel that appears in case (c). But fortunately this never leads to a condition on an unbounded number of tunnels, since in the induction those rightmost tunnels either disappear or stay.

To evaluate the complexity we use a pessimistic bound on the table size, $OPT(P((j,j),(j+1,j),l_0,l_1,[u,v]))$. The values of l_0, l_1 are polynomial since they are in bijection with the pseudo-sources locations, $j \in [1,n]$. Since $[u,v]$ is either an end or a head segment we can store it in space $2n$. Therefore we get size $\Theta(n^4)$ for the tables, and if we add the conditioning on the rightmost tunnel from the rightmost source we get $\Theta(n^5)$.

Finally, to improve the complexity we can use classic scaling technics to get space $\frac{\log n}{\varepsilon}^5$ and approximation factor $1+\varepsilon$.

6 Conclusions and Further Research

In this paper we modeled a question raised by label minimization in GMPLS networks as a hypergraph layout problem. In the single commodity case we showed the problem to be closely related to well studied VPL problems. However, the optimization criteria (average hop count and average load) that appear in our problem are ones of the less studied. We observed that approximation results follow immediately from extension of the results known for fixed depth hierarchical facility location (equivalently, bounded depth metric Steiner trees) to the average depth case. We also gave a general $\log n$-approximation that is universal (that is, it does not depend on the traffic), as well as hardness results.

In the multi-sources case, we presented a dynamic program on the path that is polynomial when the number of sources is fixed. So finding a polynomial algorithm in the general case on the path remains open; likely extensions of the dynamic program to the case of trees and bounded treewidth networks remain to be done. Last, we believe that more general approximation results can be given for low dimension Euclidean metric graphs using the classical Arora paradigm.

References

1. Bermond, J.-C., Coudert, D., Moulierac, J., Perennes, S., Rivano, H., Sau, I., Solano Donado, F.: MPLS label stacking on the line network. In: Fratta, L., Schulzrinne, H., Takahashi, Y., Spaniol, O. (eds.) NETWORKING 2009. LNCS, vol. 5550, pp. 809–820. Springer, Heidelberg (2009)
2. Bermond, J.-C., Marlin, N., Peleg, D., Pérennes, S.: Directed virtual path layouts in ATM networks. Theoretical Computer Science 291(1), 3–28 (2003)

3. Bern, M., Plassmann, P.: The Steiner problem with edge lengths 1 and 2. Information Processing Letters 32, 171–176 (1989)
4. Bhatnagar, S., Ganguly, S., Nath, B.: Creating Multipoint-to-Point LSPs for traffic engineering. IEEE Commun. Mag. 43(1), 95–100 (2005)
5. Charikar, M., Chekuri, C., Cheung, T., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation algorithms for directed Steiner problems. In: Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 192–200 (1998)
6. Gerstel, O., Wool, A., Zaks, S.: Optimal layouts on a chain ATM network. Discrete Applied Mathematics 83, 157–178 (1998)
7. Goemans, M.X., Goldberg, A.V., Plotkin, S., Shmoys, D.B., Tardos, E., Williamson, D.P.: Improved approximation algorithms for network design problems. In: Proc. of the 5th annual ACM-SIAM symposium on Discrete algorithms (SODA), pp. 223–232 (1994)
8. Khuller, S., Vishkin, U.: Biconnectivity approximations and graph carvings. Journal of the ACM 41, 214–235 (1994)
9. Ramos, F., et al.: IST-LASAGNE: Towards all-optical label swapping employing optical logic gates and optical flip-flops. IEEE J. Sel. Areas Commun. 23(10), 2993–3011 (2005)
10. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: Proc. of the 29th annual ACM Symposium on Theory of Computing (STOC), pp. 475–484 (1997)
11. Saito, H., Miyao, Y., Yoshida, M.: Traffic engineering using multiple MultiPoint-to-Point LSPs. In: Proc. of IEEE INFOCOM, pp. 894–901 (2000)
12. Solano, F., Caenegem, R.V., Colle, D., Marzo, J.L., Pickavet, M., Fabregat, R., Demeester, P.: All-optical label stacking: Easing the trade-offs between routing and architecture cost in all-optical packet switching. In: Proc. of IEEE INFOCOM, pp. 655–663 (2008)
13. Solano, F., Fabregat, R., Marzo, J.: On optimal computation of MPLS label binding for MultiPoint-to-Point connections. IEEE Trans. Commun. 56(7), 1056–1059 (2007)
14. Solano, F., Moulierac, J.: Routing in All-Optical Label Switched-based Networks with Small Label Spaces. In: Proc. of the 13th Conference on Optical Network Design and Modeling, ONDM (2009)
15. Solano, F., Stidsen, T., Fabregat, R., Marzo, J.: Label space reduction in MPLS networks: How much can one label do? IEEE/ACM Trans. Netw. (2009)

On Gossip and Populations

Marin Bertier[1], Yann Busnel[2], and Anne-Marie Kermarrec[3]

[1] INSA Rennes – France
[2] University of Rennes 1 – France
[3] INRIA Rennes – Bretagne Atlantique – France

Abstract. Gossip protocols are simple, robust and scalable and have been consistently applied to many (mostly wired) distributed systems. Nevertheless, most validation in this area has been empirical so far and there is a lack of a theoretical counterpart to characterize what can and cannot be computed with gossip protocols.

Population protocols, on the other hand, benefit from a sound theoretical framework but little empirical evaluation. In this paper, we establish a correlation between population and gossip-based protocols. We propose a classification of gossip-based protocols, based on the nature of the underlying peer sampling service. First, we show that the class of gossip protocols, where each node relies on an arbitrary sample, is equivalent to population protocols. Second, we show that gossip-based protocols, relying on a more powerful peer sampling service providing peers using a clearly identified set of other peers, are equivalent to community protocols, a modern variant of population protocols.

Leveraging the resemblances between population and gossip protocols enables to provide a theoretical framework for distributed systems where global behaviors emerge from a set of local interactions, both in wired and wireless settings. The practical validations of gossip-protocols provide empirical evidence of quick convergence times of such algorithms and demonstrate their practical relevance. While existing results in each area can be immediately applied, this also leaves the space to transfer any new results, practical or theoretical, from one domain to the other.

1 Introduction

In analogy with rumour spreading among human beings, gossip protocols provide a scalable, robust and reliable substrate for many peer to peer applications [11,13,18,22,23]. They have recently received an increased attention due to their scalability and quick convergence in large-scale dynamic settings. In a gossip protocol, each node in the system periodically exchanges information with another peer sampled from the network. The robustness of gossip protocols stems from their random flavour in the sampling. However, while there are some ad-hoc analyses available for specific protocols [11,14,16,22], most validation in the area has been so far achieved through extensive simulations and experimentations.

In [21], a generic practical gossip substrate has been defined. In this model, a protocol is defined by three functions: *(i)* **the peer selection**, identifying the gossip target,

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 72–86, 2010.

provided by a *peer sampling service*; *(ii)* the **data exchanged**, specifying the information exchanged between the peers during a gossip interaction and *(iii)* the **data processing** following an interaction. While this framework was initially defined to unify gossip membership systems, it has been shown to be generic enough to be applied for the whole spectrum of gossip protocols including reliable dissemination, distributed computation, and overlay construction [24]. Yet, there is a lack of clear theoretical framework enabling reasoning about the power and limitations of this model.

On the other hand, population protocols [1] provide theoretical foundations for distributed systems in which global behavior emerges from a set of simple interactions between their agents. Originally developed in the context of mobile tiny devices, typically sensors, in this model, agents are considered anonymous, and therefore, undistinguishable. Many variants of population protocols exist [2,4,5,6,10]. Among them, community protocols [17] augment the original model by assigning agents a unique identifier and letting nodes remember a limited number of other identifiers. Not only this significantly increases the computation power of the system but also provides a way to tolerate a bounded number of byzantine failures. In the sequel, the class of population protocols and variants will be refered as *population protocols*, and the original model as *basic population protocol*.

More specifically, the population protocol model consists in a finite space of agent's states, a finite set of inputs, a finite set of outputs and a transition function. The set of possible node's interactions is represented by a graph. When two agents are sufficiently close for a sufficiently long time, they interact by exchanging their local information, and update their state according to the transition function. For instance, if agents are small devices embedded on animals, an interaction takes place each time two animals are in the same radio range. The interaction patterns, orchestrated by a scheduler, are considered as unpredictable. Yet, the scheduler is assumed to be *fair i.e.*, it ensures that any reachable global system state can be reached infinitely often. In the absence of global knowledge, agents cannot usually verify that the protocol has terminated, therefore the model considers convergence (of the distributed output) rather than termination.

Contributions: Correlating population and gossip protocols Our contributions in this paper stem from the observation that population and gossip protocols bear many resemblances. They both rely on a *scheduler* orchestrating the interactions between nodes. The scheduler, fair by assumption in population protocols, specifies the node interactions in a mobile environment while the scheduler is a *peer sampling service* in gossip protocols providing nodes with gossip targets. Both aim at achieving an emerging global behavior from a set of local interactions in a fully decentralized manner. The main contribution of this paper is to acknowledge these similarities and leverage them in both contexts.

On one hand, the gossip structure presented in [21] provides a practical generic framework in the context of wired systems. Yet, this does not provide a fine-grained classification. Works in this area have shown that the gossip protocols scale well in practice and convergence quickly. On the other hand, population protocols provide a theoretical framework for wireless systems. They clearly define the power and limitation of such protocols. Population protocols show that such systems composed of anonymous agents ensure the convergence of a clearly defined set of functions. Community

protocols extend this model, by adding a bounded set of identifiers. However, until now, these models have not significant practical implication. In this paper, we present the following contributions:

- We establish a correlation between population and gossip protocols and introduce a *first classification* of gossip protocols depending on the nature of the underlying peer sampling service. More precisely, we identify two classes of gossip protocols: *anonymous* (AGP) and *non anonymous* (NGP).
- We show that anonymous gossip protocols are *equivalent* to basic population protocols;
- We show that non anonymous gossip protocols are *equivalent* to community protocols;
- By doing so, we leverage the theoretical framework of population protocols for understanding the power and limitations of gossip protocols. Likewise, we exploit the results obtained in the area of gossip protocols to draw conclusions on the practicality of population protocols. This enables us to provide both theoretical and practical considerations for such large-scale systems: the parallel between population and gossip protocols can be exploited for both existing and new results, as we propose in [9]. For instance, applying gossip experiments to population protocols enables to show the convergence behavior of these protocols. Likewise, applying some results from population protocols to the gossip-based protocols enable to show their computability or extract some interesting bounds as the one proposed in [9], where we provide a new result in the context of population protocol namely the *optimality of uniform distribution of interactions* [8] with respect to speed of convergence. Then, we use the equivalence property to extend it to gossip protocols. This enables us to conclude that the random peer sampling service [21] is optimal for the speed of convergence of gossip protocols. This is a clear illustration of how the correlation can be exploited. For space reasons, this last point is not developed in the paper. Details are available in [8].

2 Population vs. Gossip Protocols

2.1 Background on Population Protocols

In this section, we briefly present the basic population protocol model and the community protocol variant, which relaxes the assumption on the anonymity of agents.

Basic population protocol The basic population protocol model, initially introduced in [1], is composed of a collection of agents, interacting pairwise in an order determined by a fair scheduler. Each agent has an input value and is represented by a finite state machine. This agent can only update its state through an interaction. Updates are defined by a transition function that describes the function f computed by the system. At each interaction, the agents compute an output value from their current state and converge eventually to the correct output value, depending to the inputs initially spread to the agents.

More formally, a population protocol is composed of:

- a complete interaction graph Λ linking a set of $n \geq 2$ agents;
- a finite input alphabet Σ;
- a finite output alphabet Y;
- a finite set of possible agent's states Q;
- an input function $\iota : \Sigma \to Q$ mapping inputs to states;
- an output function $\omega : Q \to Y$ mapping states to outputs;
- a transition relation $\delta : Q \times Q \to Q \times Q$ on pairs of states.

In the following, we call $(p, q) \mapsto (p', q')$ or $(p, q, p', q') \in \delta$. A transition can occur between two agents' states only if these two agents have an interaction. The protocol is deterministic if δ is a function (*i.e.* at most one possible transition for each pair in Q^2).

A configuration of the system corresponds to an unordered multiset containing states of all agents. We denote $C \to C'$ the fact that a configuration C' can be obtained from C in one step (*i.e.* with only one transition for one existing interaction). An execution of the protocol is a finite or infinite sequence of population configurations $C_0, C_1, C_2, \ldots$ such that $\forall i, C_i \to C_{i+1}$.

As introduced above, the order of the interactions is unpredictable, and decided by the scheduler. The scheduler is assumed to be *fair*, *i.e* a feasible configuration cannot be endlessly ignored. In other words, if a configuration C appears an infinite number of times during an execution, and there exists a step $C \to C'$, then C' must also appear an infinite number of times in the execution. This ensures that any attainable configuration is eventually reached.

Community protocols. Many variants of the last model exist. In this paper, we focus on the community protocol [17] extension, which significantly increases the computational power. This model augments the basic population protocol model by assigning unique identifiers to agents. All possible identifiers and a special symbol $\perp$ are grouped in an infinite set U. The difference between basic population protocols and community protocols is the definition of the set of states: $Q = B \times U^d$ where B is the initial definition of the population protocol's set of states collapsed to a memory of d identifiers. As in population protocols, algorithms cannot use any bound on the number of agents and moreover, U is infinite. In order to maintain the population protocol spirit in this extended model, some constraints are added: only existing agent identifiers can be stored in the d slots intended for identifiers of an agent's state and no other structural information about identifiers can be used by algorithms. We consider, for $q \in Q$ and $id \in U$, that $id \in q$ means that q stores id in one of its d indentifier slots. Thus, community protocols have to verify the two following formal constraints:

- $\forall (q_1, q_2) \mapsto (q_1', q_2') \in \delta, id \in q_1' \vee id \in q_2' \Rightarrow id \in q_1 \vee id \in q_2$
- For $q = \langle b, u_1, u_2, \ldots, u_d \rangle \in Q$, let $\hat{\pi}(q) = \langle b, \pi(u_1), \pi(u_2), \ldots, \pi(u_d) \rangle$ where π a permutation of U with $\pi(\perp) = \perp$. We assume that: $\forall (q_1, q_2) \mapsto (q_1', q_2') \in \delta : (\hat{\pi}(q_1), \hat{\pi}(q_2)) \mapsto (\hat{\pi}(q_1'), \hat{\pi}(q_2')) \in \delta$.

In short, the first assumption ensures that no transition introduce new identifiers and the second one that identifiers can only be stored or compared for equality, but not manipulated in any other way. Any population protocol can be viewed as a community protocol with $d = 0$.

Finally, a population or community protocol stably computes a function $f : \Sigma^+ \rightarrow Y$ if $\forall n \in \mathbb{N}, \forall \sigma \in \Sigma^n$, every fair execution with n agents initialized with the elements of σ, eventually stabilizes to output $f(\sigma)$. That means that the output value of every agent eventually stabilizes to $f(\sigma)$.

2.2 Gossip Protocols: A Practical Framework

Originally introduced for information dissemination, gossip protocols are simple, robust and scalable. Initially, epidemic-based algorithms have been proposed for database maintenance in [11]. Since then, they have been applied in many settings in wired and wireless systems as in [7,13,14,15,18,26,27,28].

A generic framework has been proposed in [21] to provide a generic substrate for gossip peer sampling protocols, providing a common ground for membership systems. In this paper, the authors explore the resulting topologies and show that the parameters of the generic gossip protocols can be set to achieve random-like graph topologies *i.e.* providing each node with a random sample of the network. This has been achieved through extensive experimentations and has been recognized as a way to achieve random peer sampling in large-scale dynamic networks.

In this framework, each peer maintains a local view of size c of the system, representing its restricted knowledge of the systems. The peer sampling service provides a sample from that view. This framework relies on three basic functions:

SelectPeer() returns a peer from the local view. This function is used to select the gossip target;

DataExchange() returns the data to be exchanged over a gossip communication;

DataProcessing() returns the resulting state and specifies the way the data exchanged are processed.

Each peer runs an active and a passive threads (see Algorithm 1). The active thread is run periodically and launches a gossip interaction. A gossip target is selected from its local view using (*SelectPeer()*), data is exchanged (*DataExchange()*) and processed between the two interacting peers (*DataProcessing()*). Concurrently, in a passive thread, the selected node, after reception, sends its own information and also processes the received information.

Algorithm 1. Generic Gossip Protocol

Active thread
Do once for each T time units at a random time
begin
  p = SelectPeer()
  Send DataExchange($state$) **to** p
  Receive $info_p$ **from** p
  $state$ = DataProcessing($info_p$)
end

Passive thread
Do forever
begin
  Receive $info_q$ **from** q
  Send DataExchange($state$) **to** q
  $state$ = DataProcessing($info_q$)
end

Initially proposed in the context of protocols achieving unstructured topologies, it turns out that this very substrate is generic enough to be used for many other purposes [24]. For example, message dissemination [23,13] can be achieved by parameterizing the protocol as follows. *(i) SelectPeer()* should return a random peer; *(ii) DataExchange()* should contain the message to disseminate; and *(iii) DataProcessing()* should do nothing. Likewise, distributed computations (average, sum or quantile) [20,22], gossip size estimation [12,25] or overlay construction [18,28] can be achieved using the same protocol. This substrate provides a general framework for practical implementations of gossip protocols. Yet, to the best of our knowledge, there is no theoretical counterpart (in the sense of a framework).

3 A Classification of Gossip Protocols

3.1 On the Power of the Peer Sampling Service

In this section, we propose a novel classification of gossip protocols stemming from the observation that, although studied independently by two different communities, population and gossip protocols have a lot in common. Both class of protocols rely on the following properties:

- a fully decentralized model;
- a set of agents, having a finite storage capacity, periodically interacting in a pairwise manner. The agents are mobile and communicate in a wireless manner in population protocols; they are static and communicate through a dynamic network on a fixed infrastructure in gossip protocols;
- an unpredictable order of interactions orchestrated by a fair scheduler in population protocols modelling the agents' mobility patterns and by a peer sampling service serving the *selectPeer()* function in gossip protocols;
- a function specifying the way data is processed over an interaction: this is the transition function δ in population protocols and the *DataProcessing* function in gossip protocols;
- a state exchange over an interaction: this is the state value in Q in population protocols and the *DataExchanged* function in gossip protocols.

Considering the gossip protocols in this light, we were able to make a parallel between *(i)* the difference between the basic population and community protocols and *(ii)* the requirements for peer identifiers in gossip protocols.

More specifically, gossip protocols differ from their requirement with respect to peer anonymity. This is instantiated by the nature of the underlying peer sampling protocol. The peer sampling service, *i.e.* the "black box" providing a peer with a given sample of the network, may either return any sample for the implementation of the gossip protocol. Therefore, we introduce the following classification. Two main classes of gossip protocols can then be defined depending on the power of the underlying peer sampling service with respect to anonymity requirements.

AGP: Anonymous Gossip protocols do not require being aware of the identities of any peer for any of the three functions of the generic protocol. This is typically the case of protocols achieving some simple distributed computations such as average computation [20,22] or system size estimations [19,25]. Gossip dissemination protocols where each peer gossips to k nodes picked uniformly at random also fall into this category. Such protocols only rely on a peer sampling service providing them with a sample of the network, be it random or biased [7,23].

NGP: Non-anonymous Gossip Protocols are not oblivious to the identities of peers they are communicating with or any other. Typically, gossip overlay construction protocols fall into this class. The identities of peers are required in the three functions of the substrate aforementioned. Non-anonymous gossip protocols have been used to implement overlays ranging from unstructured networks, providing random-graph like topologies [21] to structured networks [18,23].

3.2 Between Synchronous and Asynchronous

To refine the classification, we take into account the two main models of communication channels. In a synchronous model, message delay, clock drift and time required to execute an algorithm step are bounded and these bounds are known. On the contrary, in an asynchronous model, there is no bound. Gossip protocols have a periodic behavior, in which at each step, every agent launches an exchange with another agent. In a synchronous system, the gossip keeps its periodic behavior, and so every agent is in the same communication round. In an asynchronous system, the clock drift and the fact that a time required by a gossip exchange cannot be bound infers that the periodic behavior is lost.

3.3 On the Computational Power of Gossip Protocols

Enriching our model with the communication synchronism property, we obtain a refined classification as the one presented earlier. We now classify gossip protocols in four classes:

syncAGP Synchronous Communication and Anonymous Nodes;
asyncAGP Asynchronous Communication and Anonymous Nodes;
syncNGP Synchronous Communication and Non-anonymous Nodes;
asyncNGP Asynchronous Communication and Non-anonymous Nodes.

Obviously, the power of non-anonymous gossip protocols is greater than anonymous ones as the use of node identifier enables to achieve distributed computations which are impossible in the anonymous context (*eg.* exponential computation, logical overlay construction, *etc*.). Thus, we have:

$$\text{asyncAGP} \prec \text{asyncNGP} \quad \text{and} \quad \text{syncAGP} \prec \text{syncNGP}$$

As far as anonymous gossip protocols are concerned, it is possible to leverage the periodicity of exchange in order to increase their computational power. For instance, it is possible in syncAGP to establish a global time clock, thanks to the cycle structure, but

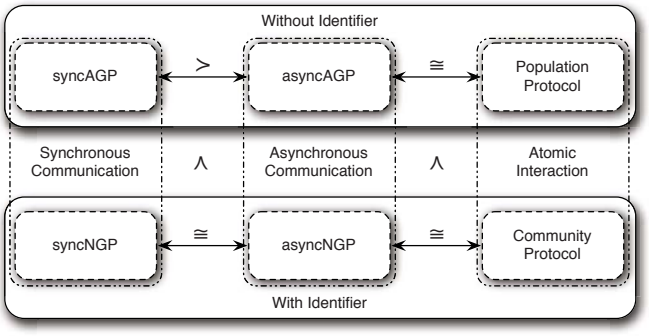


Fig. 1. Relationship between gossip-based and population protocols

not in asyncAGP (due to the unbounded message delay). Then, it is obvious to conclude that:

$$\text{asyncAGP} \prec \text{syncAGP}$$

Finally, in the NGP context, we raise in Remark 1 that the identification of nodes enables to emulate synchronous communications, such that:

$$\text{asyncNGP} \cong \text{syncNGP}$$

This classification and the relation between all the considered models are summarized in Figure 1. This provides a refined classification of gossip protocols based on the synchronism and anonymity properties. Yet, there is no formal framework to define what can and cannot be achieved with the susmentioned protocols. Establishing the parallel between population and gossip protocols provides a first answer to that question.

4 Bridging the Gap between Population and Gossip Protocols

In a nutshell, in AGP, a peer sampling service provides each peer with another peer to communicate with, regardless of its identifier. If the peer sampling service ensures that any pairwise interaction can endlessly take place, then a protocol of AGP resembles a basic population protocol. Inversely, a protocol from NGP requires a peer sampling service to provide each node with a set of clearly identified peers, potentially along with more information about each peer, where the identifier (whether it is an identifier or an IP address) is crucial. This means that the *SelectPeer* or the *DataProcessing* use the identifier or information attached to specific peer to achieve a given functionality. This clearly matches the community protocol model described above. We claim that these resemblances are actually equivalences and provide the proofs in this section as illustrated on Figure 1.

4.1 Equivalence between Basic Population and Anonymous Asynchronous Gossip Protocols

In this section, we prove that the basic population and anonymous asynchronous gossip protocols (asyncAGP) are equivalent.

Theorem 1. *A predicate is computable by a basic population protocol if and only if it can be computed by an anonymous gossip protocol via an asynchronous communication model (asyncAGP).*

Proof. In order to prove the equivalence, we consider the functions computable by basic population protocols and asyncAGP. Then, we prove in Lemmas 1 and 2 that they belong to the same equivalence class. In fact, we prove below that the class of functions computable by a basic population protocol is a subset of the ones computable by asyncAGP, and *vice-versa*. □

On one hand, consider that PP $\prec$ asyncAGP with the following lemma.

Lemma 1. *If f is computable by a basic population protocol, then there exists a protocol from asyncAGP which can compute f.*

Proof. Let $\mathcal{P}$ the basic population protocol computing f and defined by the 7-tuples $(\Lambda, \Sigma, Y, Q, \iota, \omega, \delta)$. Consider the anonymous gossip protocol $\mathcal{G}$ described below. We have to show that $\mathcal{G}$ simulates $\mathcal{P}$.

Similarities: Each agent in Λ is hosted by a specific peer of $\mathcal{G}$. Input, output and state sets are the same in $\mathcal{G}$ than in $\mathcal{P}$. *A fortiori*, both map function ι and ω remain identical in $\mathcal{G}$.

Dealing with the transition function: The $\mathcal{G}$'s *DataProcessing* function is defined from δ: consider two peers in the system l and r, gossiping with each other at a time t. Let assume that l initiates the gossip exchange with r. Let p_l (respectively p_r) the selected information obtained by *DataExchange* from l's state (respectively r's state). Thus, as l calls the function during its active thread, *DataProcessing* returns locally the third entry of the 4-tuple $(p_l, p_r, p'_l, p'_r) \in \delta$, and the state of l becomes p'_l. On the remote peer r, a call to the function *DataProcessing* during its passive thread returns the last entry of the same 4-tuple (p_l, p_r, p'_l, p'_r).

Thus, the sequence of population configurations is valid and represents the basic population protocol $\mathcal{P}$, as it only stems from the transition function δ using pairwise interactions.

On the fairness assumption: The last assumption to verify is the fairness condition. In asyncAGP, the scheduler is fully defined by the order of gossip exchanges, itself defined by *(i)* the *selectPeer* function; *(ii)* the randomization of the gossip time and; *(iii)* the asynchronous environment which, as we mentioned before, potentially leads to gossip exchange losses. In that context, every possible finite scheduling has a non-null probability to happen. Thus, every possible transaction between two system configurations $C \rightarrow C'$ has a non-null probability to happen. Moreover, if the asynchronism acts against this condition, given a specific interaction, which is avoided for a while, the probability that it continues to be avoided tends to zero. So, if the configuration C appears an infinite number of times during an execution of $\mathcal{P}$ in the aforementioned context, then C' also appears an infinite number of times in this execution. The fairness assumption is then verified.

Then, $\mathcal{G}$ simulates the basic population protocol $\mathcal{P}$, which computes the function f. Thus, for any function computable by basic population protocols, there exists an anonymous gossip protocol, which stably computes this function. □

On the other hand, let's show the inverse of Lemma 1, corresponding to the second implication of Theorem 1: asyncAGP $\prec$ PP.

Lemma 2. *If f is computable by a protocol from asyncAGP, then there exists a basic population protocol which can compute f.*

Proof. Let $\mathcal{G}$ an anonymous gossip protocol that computes a specific function f using the primitive *DataExchange* and *DataProcessing*. As presented above, in a gossip protocol, peers are modeled by a finite-state machine.

Mapping the domain of the transition function: The domain of *DataProcessing* is finite (and corresponds to the Cartesian product of $\mathcal{D}_S$, the set of peer states, with $\mathcal{D}_E$, the range of *DataExchange*). Moreover, as *DataProcessing* is a function, its range is also finite by definition. Based on these sets, we define $\mathcal{D}_\mathcal{G}$ a specific subset of the Cartesian product between the domain and the codomain of *DataProcessing* (*i.e.* $\mathcal{D}_\mathcal{G}$ contains each ordered pair such that the first entry is in the domain of *DataProcessing* and the second entry is the mapped element of this first entry by *DataProcessing*). More formally, $\mathcal{D}_\mathcal{G} \subseteq (\mathcal{D}_S \times \mathcal{D}_E) \times \mathcal{D}_S$. Thus, $\mathcal{D}_\mathcal{G}$ is finite and contains all the possible transitions of peer states, based on the knowledge of a remote peer sub-state.

Design of the basic population protocol for the purpose of simulation: Consider the following basic population protocol $\mathcal{P}$, represented by the 7-uplet $(\Lambda, \Sigma, Y, Q, \iota, \omega, \delta)$. Consider a complete interaction graph Λ. Let the set of agent states be identical to the set of peer states, *i.e.* $Q = \mathcal{D}_S$. Consider that Σ and Y are the same as the input and output sets of $\mathcal{G}$, if they exist. In this case, ι and ω are the same functions than the ones which respectively associate the input set of $\mathcal{G}$ to $\mathcal{D}_S$, and $\mathcal{D}_S$ to the output set of $\mathcal{G}$. Conversely, if no specific input and output sets are defined in $\mathcal{G}$, then $\Sigma = Y = \mathcal{D}_S$ and $\iota \equiv \omega$ corresponding to the identity function. Finally, the transition function δ is defined as follows.

$$\forall (s_l, s_r, s_l') \in \mathcal{D}_\mathcal{G}, \exists (s_r, s_l, s_r') \in \mathcal{D}_\mathcal{G} \quad \text{such that} \quad (s_l, s_r, s_l', s_r') \in \delta.$$

On the periodicity of the fair scheduler: Periodicity of exchange is an inherent characteristic of many gossip protocols. However, the only difference between asyncAGP and syncAGP is that asyncAGP potentially temporary jeopardize the periodicity of exchanges, in case of arbitrary long transmission delays. Thus, as presented in Lemma 1, no periodic assumption can be considered in an asynchronous environment. Therefore, a fair scheduler is sufficient to lead to a correct execution of $\mathcal{G}$, using the aforementioned $\mathcal{P}$.

Thus, there exists a basic population protocol $\mathcal{P}$, which simulates the considered asyncAGP $\mathcal{G}$ and computes the function f. Then, for any function computable by a protocol from asyncAGP, there exists a basic population protocol, which stably computes this function. □

4.2 Equivalence between Community Protocols and NGP

Along the same lines, we prove here the following theorem in order to prove the equivalence between community and NGP protocols. In fact, we show in the proof of Lemma 4

that it is possible to simulate the periodicity of protocols from syncNGP using a protocol from asyncNGP (these two classes are then equivalent).

Theorem 2. *A predicate is computable by a community protocol if and only if it can be computed by a non-anonymous gossip protocol (NGP).*

Proof. As Theorem 1, the proof of Theorem 2 is directly infered from the statements of Lemmas 3 and 4, which show respectively both implications of this equivalence. □

Consider the first implication of this theorem. Inspired from the equivalence between basic population protocols and asyncAGP, the following theorem is almost trivial.

Lemma 3. *For each f computable by a community protocol, there exists a NGP protocol which compute f.*

Proof. The only difference between a basic population protocol and a community protocol consists in the definition of the set of states (*i.e.* $Q = B \times U^d$) and the two constraints on the state's identifier part (*i.e.* the part belonging to U^d cannot be used freely). Then, due to Lemma 1 and its sketch of proof, the protocol from NGP has to be designed to simulate a given community protocol $\mathcal{C}$. Then, we can still consider the function *selectPeer* as a black box which provides a fair scheduler. In other hand, functions *DataExchange* and *DataProcessing* are defined respectively on the domain $\mathcal{D}_S = B \times U^d$ (instead of B in the anonymous gossip version) and $\mathcal{D}_S \times \mathcal{D}_E$.

This does not violate the definition of NGP as the additional information used here only depends on the presence of unique identifier on agents, which is a mandatory assumption in non anonymous gossip protocols. □

Finally, let now show the opposite of Lemma 3, corresponding to the second part of Theorem 2.

Lemma 4. *For each f computable by a NGP protocol, there exists a community protocol which computes f.*

Proof. Let $\mathcal{G}$ the given protocol from NGP. Thus, each peer in the system is aware of its unique identifier. Consider the following community protocol $\mathcal{C}$.

Preliminary assumptions on $\mathcal{C}$: We assume that, in the community protocol used on $\mathcal{C}$, agents are uniquely identified, and a unique agent is assigned a specific identifier $id_{\mathcal{L}}$. This agent is considered as the leader by all other agents. In this specific community protocol, we assume that this leader is aware of the size of the system[1] (denoted n in the sequel). In other words, the view of a peer is modeled as the set of $d-1$ identifiers in the community protocol model (one space of the d-tuple in U^d is set aside for storing its own identifier).

Summary of agent state requirement: In addition of the gossip peer state in $\mathcal{D}_S$, each agent maintains:

[1] This assumption is only expected for the synchronisation barrier. It can be relaxed using the probabilistic clock phase mechanism proposed for population protocols in [2].

- a binary variable gc_{parity} to memorize the current gossip cycle parity;
- a ternary variable gc_{progress} storing the gossip state of an agent at the corresponding cycle. gc_{progress} can only take one of the three following values: *(i)* $todo$ if the active thread has not been run yet during the current gossip cycle, *(ii)* $done$ if it has been run or *(iii)* $wait$ representing the inter-cycle state, as explained below (*i.e.* the agent waits to pass across the synchronization barrier);
- an agent's identifier variable id_{next}, which stores the identifier of the next agent to gossip with;
- the leader agent $\mathcal{L}$ maintains a counter gc_{count} used in the synchronization cycle process.

To make a long story short, the set of agent states is defined as

$$Q = \mathcal{D}_S \times \{true, false\} \times \{todo, done, wait\} \times U \times [1; n] \times U^d$$

i.e. the Cartesian product between *(i)* the domain of the function *DataProcessing* (to represent the gossip peer state), *(ii)* the domain of gc_{parity}, *(iii)* the domain of gc_{progress}, *(iv)* the identifier set U for id_{next}, *(v)* the domain of gc_{count} and finally *(vi)* U^d for the view of the agent. At initialization, each agent sets gc_{parity} to $false$, gc_{progress} to $todo$ and id_{next} to $id_{\mathcal{L}}$. Moreover, gc_{count} is set to 0 for the leader agent $\mathcal{L}$.

Simulating a gossip cycle in $\mathcal{C}$: We now describe the behavior of an agent during a gossip cycle, according to its gc_{progress} value. In the case that $gc_{\text{progress}} = todo$, the corresponding agent has not run its active thread in a given gossip round. Then, it waits to meet its next gossip partner, corresponding to the identifier stored in id_{next}. For each interaction (id_1, id_2), the agent corresponding to id_1 verifies if $gc_{\text{progress}} = todo$. If this is the case, it checks if $id_2 = id_{\text{next}}$. Only in that case, id_1 and id_2 run respectively *DataProcessing*(q_1,*DataExchange*(q_2)) and *DataProcessing*(q_2,*DataExchange*(q_1)) (where q_1 and q_2 represents respectively the state of these agents). All the possible transitions are included in $\mathcal{D}_{\mathcal{G}}$ introduced in the proof of Theorem 2. At the end of this interaction, id_1 and id_2 have updated their own state according to the previous one (q_i) and the remote one (*DataExchange*(q_j)). Finally, id_1 sets gc_{progress} to $done$. In other words, each agent waits until it encounters the agent with the identifier stored in id_{next} to gossip with.

How to simulate the T periodicity: In order to simulate the cycle in the context of community protocol, the T time slot can be simulated through a synchronization barrier. In the rest of this proof, we present how to establish such a barrier and how to assign agents to gossip cycles. So, we introduce the behavior of agents in case that $gc_{\text{progress}} \neq todo$.

Consider an agent id. After its own active gossip, the gossip state of id becomes $done$. In this state, it only waits until it encounters the agent $id_{\mathcal{L}}$. During its next interaction with $id_{\mathcal{L}}$, id sets its gc_{progress} to $wait$ and $id_{\mathcal{L}}$ increments gc_{count} by 1. Thus, all agents eventually stabilize to the $wait$ state, and gc_{count} eventually converges to n (the number of agents in the population). At this point, all agents have reached the synchronization barrier.

After the barrier, $id_{\mathcal{L}}$ enables all agents to begin the next gossip cycle as described hereinafter. $id_{\mathcal{L}}$ switches its own gc_{parity} to its opposite value, gc_{progress} to the $todo$ value, id_{next} to the returned value of *selectPeer* and finally gc_{count} to 0. At this point, each

time an agent in the $wait$ state interacts with an agent owning the opposite value of gc_{parity}, will fall into the next cycle by switching gc_{parity}, and setting gc_{progress} and id_{next} respectively to the $todo$ value and the returned value of *selectPeer*. Then, all agents eventually leave the $wait$ state of the last cycle, and are ready for their next gossip exchange.

In conclusion, if $\mathcal{G}$ computes the function f, then the community protocol $\mathcal{C}$ simulates the behavior of $\mathcal{G}$ and also computes the function f. □

Remark 1. Both classes of non-anonymous gossip protocol are equivalent.

Proof. The last step of the latter proof lets us show that a protocol from NGP in an asynchronous environment is able to simulate a syncNGP. It is obvious that asyncNGP $\prec$ syncNGP (for the same reason that in AGP – *cf.* Section 3.3). Thus, we can conclude that asyncNGP $\cong$ syncNGP, and consequently, that community protocols are equivalent to all protocols from NGP (asyncNGP $\cup$ syncNGP). □

We then have proved all the claims presented in Figure 1.

4.3 Leveraging the Relations

The equivalences above are of the utmost importance in the area of gossip protocols. They clearly define what can be computed with an anonymous gossip protocol. They show that the functions of the Presburger arithmetic eventually converge using an anonymous gossip protocol. They also prove that no other function can be computed with such a protocol [1,3]. This is a new and important result in the area of gossip protocols. On the other hand, the empirical results on the convergence times and practicality of the gossip protocols can be used to evaluate the efficiency of population protocols [9].

Likewise, gossip protocols relying on a peer sampling service providing peers, with a bounded set of identifiers, are equivalent to community protocols. This can be used to achieve any computation of symmetric function from $NSPACE(n \log n)$ (namely a high number of functions) and also to implement algorithms tolerating failures (and not only benign ones).

These results can be leveraged for existing results as well as results to come in both areas. Due to space constraint, we cannot develop these observations in this paper, but in [9], we illustrate this claim by considering a gossip protocol and considering it from the population protocol standpoint and the other way around.

5 Conclusion and Future Works

The main contribution of this paper is to establish a correlation between population and gossip protocols. This parallel between two worlds, explored so far independently, offers several extremely interesting outcomes. First it enables to provide a first classification of gossip protocols, a theoretical framework for such protocols, allowing to specify in a formal way what can and cannot be computed by a gossip protocol depending on the nature of the underlying peer sampling service. If a gossip protocol relies on

a peer sampling service oblivious to identifiers, it is equivalent to a population protocol. If the peer sampling service is identifier-aware, a gossip protocol is equivalent to a community protocol. For example, it is now clear that the multiplication, which does not belong to the Presburger arithmetic, cannot be achieved by an anonymous gossip protocol.

Conversely, this equivalence enables to leverage the properties obtained empirically on gossip protocols with respect to scalability, practicality and speed of convergence and apply them to population protocols. Apart from exploiting the already known results, this opens the door to leverage any new result in one of these two areas as our case studies demonstrate [9].ssip-based computations.

Part of the future work is to explore further the classes of population protocol to refine our classification of gossip protocols. Quantifying formally the convergence times of such protocols also remains an open issue.

Acknowledgment. We would like to warmly thank Davide Frey, Carole Delporte-Gallet and Rachid Guerraoui for theirs comments and suggestions.

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing (Special Issue: PODC 2004) 18(4), 235–253 (2006)
2. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distributed Computing (Special Issue: DISC 2007) 21(2), 183–199 (2008)
3. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: 25th annual ACM Symposium on Principles of Distributed Computing (PODC 2006), August 2006, pp. 292–299 (2006)
4. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing (Special Issue: PODC 2006) 20(4), 279–304 (2007)
5. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing population protocols. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 103–117. Springer, Heidelberg (2006)
6. Aspnes, J., Ruppert, E.: An introduction to population protocols. Bulletin of the European Association for Theoretical Computer Science, Distributed Computing Column 93, 98–117 (2007)
7. Busnel, Y., Bertier, M., Fleury, E., Kermarrec, A.-M.: GCP: Gossip-based code propagation for large-scale mobile WSN. In: The First International Conference on Autonomic Computing and Communication Systems (Autonomics 2007) (October 2007)
8. Busnel, Y., Bertier, M., Kermarrec, A.-M.: On the Impact of the Mobility on Convergence Speed of Population Protocols. Research Report RR-6580, INRIA, Rennes, France (July 2008)
9. Busnel, Y., Bertier, M., Kermarrec, A.-M.: Bridging the Gap between Population and Gossip-based Protocols. Research Report RR-6720, INRIA, Rennes, France (November 2008)
10. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: When birds die: Making population protocols fault-tolerant. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 51–66. Springer, Heidelberg (2006)
11. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: 6th ACM Symposium on Principles of Distributed Computing (PODC 1987) (1987)

12. Dionysios, K., Psaltoulis, D., Gupta, I., Birman, K., Demers, A.: Active and passive techniques for group size estimation in large-scale and dynamic distributed systems. Elsevier Journal of Systems and Software 80, 1639–1658 (2007)
13. Eugster, P.T., Handurukande, S., Guerraoui, R., Kermarrec, A.-M., Kouznetsov, P.: Lightweight probabilistic broadcast. ACM Transactions on Computer Systems 21(4), 341–374 (2003)
14. Eugster, P.T., Guerraoui, R., Kermarrec, A.-M., Massoulié, L.: Epidemic information dissemination in distributed systems. IEEE Computer 37(5), 60–67 (2004)
15. Gavidia, D., Voulgaris, S., van Steen, M.: Epidemic-style monitoring in large-scale wireless sensor networks. Technical Report IR-CS-012, Vrije Universiteit Amsterdam (2005)
16. Georgiou, C., Gilbert, S., Guerraoui, R., Kowalski, D.R.: On the complexity of asynchronous gossip. In: 27th annual ACM Symposium on Principles of Distributed Computing (PODC 2008), August 2008, pp. 135–144 (2008)
17. Guerraoui, R., Ruppert, E.: Even small birds are unique: Population protocols with identifiers. Technical Report CSE-2007-04, Dept of Computer Science and Engineering, York University (September 2007)
18. Jelasity, M., Babaoglu, O.: T-Man: Fast gossip-based construction of large-scale overlay topologies. Technical Report UBLCS-2004-7, University of Bologna, Department of Computer Science, Bologna, Italy (May 2004)
19. Jelasity, M., Kermarrec, A.-M.: Ordered slicing of very large-scale overlay networks. In: 6th IEEE International Conference on Peer-to-Peer Computing (P2P 2006), September 2006, pp. 117–124 (2006)
20. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems 23(3), 219–252 (2005)
21. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3), 8 (2007)
22. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003), October 2003, pp. 482–491 (2003)
23. Kermarrec, A.-M., Massoulié, L., Ganesh, A.J.: Probabilistic reliable dissemination in large-scale systems. IEEE Transactions on Parallel and Distributed Systems 14(3) (March 2003)
24. Kermarrec, A.-M., van Steen, M. (eds.): ACM Operating Systems Review on Gossip Potocols 41(5) (October 2007)
25. Massoulié, L., Le Merrer, E., Kermarrec, A.-M., Ganesh, A.: Peer counting and sampling in overlay networks: random walk methods. In: 25th ACM Symposium on Principles of Distributed Computing (PODC 2006), July 2006, pp. 123–132 (2006)
26. van Renesse, R.: Power-aware epidemics. In: International Workshop on Reliable Peer-to-Peer Systems (2002)
27. Simonton, E., Kyu Choi, B., Seidel, S.: Using gossip for dynamic ressource discovery. In: 35th International Conference on Parallel Processing (ICPP 2006), August 2006, pp. 319–328 (2006)
28. Voulgaris, S., Gavidia, D., van Steen, M.: Cyclon: Inexpensive membership management for unstructured P2P overlays. Journal of Network System Management 13(2) (2005)

Reconstructing Visibility Graphs with Simple Robots

Davide Bilò[1], Yann Disser[1], Matúš Mihalák[1], Subhash Suri[2,⋆], Elias Vicari[1], and Peter Widmayer[1]

[1] Institute of Theoretical Computer Science, ETH Zürich
{dbilo,ydisser,mmihalak,vicariel,widmayer}@inf.ethz.ch
[2] Department of Computer Science, University of California, Santa Barbara
suri@cs.ucsb.edu

Abstract. We consider the problem of finding a minimalistic configuration of sensors that enable a simple robot inside an initially unknown polygon $\mathcal{P}$ on n vertices to reconstruct the visibility graph of $\mathcal{P}$. The robot can sense features of its environment through its sensors, and it is allowed to move from vertex to vertex.

We aim at understanding which sensorial capabilities are sufficient for the reconstruction of the visibility graph of $\mathcal{P}$. We are able to show that the *combinatorial visibilities* at every vertex do not contain enough information even when combined with the knowledge of the exact interior angle at each vertex. Using sensors that can put distant vertices into a spatial relation on the other hand can in some cases enable our robot to reconstruct the visibility graph of $\mathcal{P}$. We show that this is true for a sensor that can distinguish whether the angle between two vertices the robot sees is convex or reflex, as long as the robot is capable of identifying the vertex it last visited. We also show that measuring angles exactly is enough, if the robot has a compass.

1 Introduction

We aim at finding minimalistic motor and sensory capabilities that enable simple robots to explore an unknown environment. The exploration of environments is an important robotic task [8]. Recently, it has also been studied in the context of simple robots giving rise to different modelling approaches [4,10,11]. We model robots as points in an initially unknown polygonal environment $\mathcal{P}$ whose number of vertices n is assumed to be known. We allow a robot to collect sensory input while it is located at a vertex, and to move from its current vertex to any vertex that it sees. In the spirit of keeping robots simple, our robots in particular cannot sense while they move. Our basic sensing capability allows each robot to see all vertices that are visible from its current position, in counter-clockwise (ccw) order, where a vertex is said to be visible by a robot sitting on another vertex if the line segment connecting both vertices lies entirely in $\mathcal{P}$. Vertices have no

⋆ This author wishes to acknowledge the support provided by the National Science Foundation under grants CNS-0626954 and CCF-0514738.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 87–99, 2010.

characteristics that identify them globally; they can be distinguished only in a local sense by the relative position in ccw order when looking from some other vertex. For a variety of configurations of additional sensory capabilities, we analyze whether a robot is capable to infer the visibility graph of $\mathcal{P}$. Recall that the visibility graph consists of a vertex for each polygon vertex, with an edge between two vertices if the polygon vertices are mutually visible. The characterization of visibility graphs and their reconstruction from polygon geometry have been studied extensively [5]. We are interested in the problem of deciding whether a given set of sensory and motor capabilities is powerful enough to allow a robot to reconstruct the visibility graph of its polygonal environment without any prior knowledge of the polygon's geometry.

An earlier study [10] assumes that robots have no notion of and no way of measuring coordinates, distances or angles. Instead, these robots are limited to distinguish whether any two visible vertices are neighbors on the polygon boundary. This concept is usually referred to as *combinatorial visibility* and will be defined more formally below. In a convex polygon, for instance, a robot at any of the vertices sees all other vertices and sees that any two consecutive vertices in cyclic order are neighbors (this is obviously not true in any other polygon). There was hope that the knowledge of all combinatorial visibilities might be enough for a robot to derive the visibility graph of $\mathcal{P}$. In this paper, we show that this knowledge alone is in fact not sufficient. In fact our result implies that a robot that senses combinatorial visibilities cannot reconstruct the visibility graph, if it only moves along the boundary. The question whether the robot is capable of reconstructing the visibility graph, if allowed to move to any vertex it sees remains open.

For certain robotic tasks, such as the rendezvous of two robots in an unknown polygon, a visibility graph might not be needed, for instance if the simple polygon looks non-periodic to the robot(s). For periodic-looking polygons, there was hope that the vertices seen from a given vertex and those seen from a "periodic partner" of the given vertex (details follow in Section 3) would themselves be periodic partners; this property would have allowed a variety of tasks to be solved. We show that this, unfortunately, is not the case.

The question arises what kind of minimal information a robot needs to make the derivation of the visibility graph possible. We show that adding the knowledge of all the inner polygon angles to the knowledge of combinatorial visibilities is still not enough. Instead, we equip the robot with sensors that are able to put the vertices a robot sees into a certain spatial relation only. One example of such a sensor distinguishes whether the angle between any pair of vertices the robot sees is convex or reflex. We show that if we add the ability of the robot to know were it came from when moving from vertex to vertex, this simple sensor is already sufficient. We also show that sensing exact angles is sufficient as long as we add a compass that provides a global reference direction.

Related Work. The capabilities of robots and strategies for different robotic tasks have been studied in a broad variety of settings [8]. Our setting of simple robots in a polygonal environment was first introduced in [10].

While we focus on mapping unknown environments, other robotic tasks have been studied using simple robots. One example is the gathering problem in the plane with multiple robots [3]. Examples in polygonal environments include localisation problems [7] and the construction of competitive watchman tours [6]. In contrast to our approach, the models used mostly allow robots to sense continuously while moving.

There have also been other results in the field of mapping unknown environments, again many focus on robots that perceive their surroundings continuously [9]. Mostly the aim is not to reconstruct combinatorial properties of the surroundings, but rather the exact geometrical layout. More strongly related to our setting is the mapping of graphs [1,2], where robots have discrete motion and sensing capabilities similar to our model. But while this problem is more general than the task of reconstructing the visibility graph, it was shown to be unsolvable for general environments without the ability to mark visited vertices.

2 Notation

In this work we consider simple polygons only. We denote the n vertices of a (simple) polygon $\mathcal{P}$ by $V=\{v_0, v_1, \cdots, v_{n-1}\}$, ordered along the boundary in counterclockwise (ccw) order. The polygon has a set of n edges $E = \{e_0, e_1, \ldots, e_{n-1}\}$, where $e_i = (v_i, v_{i+1})$, $i = 0, \ldots, n-1$. Note that from now on all primitive operations on vertex and edge indices are modulo n. In addition we assume general position, i.e. no three vertices are allowed to lie on a line.

Definition 1. *Two vertices $v_i, v_j \in V$ form a* visible pair *in $\mathcal{P}$, if the line segment $\overline{v_i v_j}$ lies entirely within $\mathcal{P}$ (in particular, v_i forms a visible pair with itself for any i). We say v_i and v_j* see each other *and write $v_i \leftrightarrow_{\mathcal{P}} v_j$. We drop the index $\mathcal{P}$ and simply write '$\leftrightarrow$', if the corresponding polygon $\mathcal{P}$ is clear from the context. We say a robot at vertex u* sees *vertex v_i, if $u \leftrightarrow v_i$.*

Definition 2. *We define* view(v_i) *of vertex v_i in $\mathcal{P}$, the* view *of vertex v_i, to be the set of vertices that v_i sees in $\mathcal{P}$. Formally,*

$$\text{view}(v_i) := \{\, v_j \in V \,|\, v_i \leftrightarrow v_j \,\}.$$

We write $\text{view}_j(v_i)$ to denote the j-th vertex, $j \geq 0$, that v_i sees in ccw order, starting at v_i itself, both $\text{view}_0(v_i)$ and $\text{view}_{|\text{view}(v_i)|}(v_i)$ denoting v_i. The *view* of a robot at vertex v_r is the view of v_r and we simply write *view* if the corresponding robot and its position are clear from the context. Similarly, we write view_i to denote $\text{view}_i(v_r)$.

When presenting algorithms for a robot we make use of a specific operation: The operation '**move to** i' moves the robot to the vertex view_i. If the robot is equipped with the corresponding sensor, it may also support the operation '**look back**' in which the robot determines the index b such that view_b is the vertex it came from during the previous **move to** operation. For other sensors we do not define operations explicitly, but rather let the robot access the measured information directly.

In the next section we introduce "combinatorial visibility" which allows robots to sense for every pair of vertices of the polygon $\mathcal{P}$ in its view, whether those vertices are neighbors on the boundary of $\mathcal{P}$. We show that the knowledge of all combinatorial visibilities is not sufficient for the reconstruction of the visibility graph of $\mathcal{P}$, even when combined with the knowledge of the exact interior polygon angle at every vertex.

In Section 4 we focus on sensors that can put distant vertices into spatial relation. One such sensor is able to measure the exact angle between the lines connecting the position of the robot with any two vertices in sight. Even when the angle measurement is not precise and the sensor can only distinguish between convex and reflex angles, we show that the visibility graph can be inferred, if we allow the robot to look back. In addition, we show that measuring exact angles is enough, if we give the robot a compass that provides a global reference direction.

3 Combinatorial Sensors

The *combinatorial visibility* of a vertex v_i is given by a binary vector who's j-th element encodes whether the j-th visible vertex and the $(j+1)$-th visible vertex form an edge of $\mathcal{P}$ or not; we call this a *combinatorial visibility vector* $\mathrm{cvv}(v_i)$. The following definitions capture this more formally. Consult Fig. 1 along with the definitions.

Definition 3. *The* combinatorial visibility vector $\mathrm{cvv}(v_i) \in \{0,1\}^{|\mathrm{view}(v_i)|}$ *of vertex $v_i \in V$ is a binary vector with the j-th element, $j \geq 0$, given by*

$$\mathrm{cvv}_j(v_i) = \begin{cases} 1, \text{ if } (\mathrm{view}_j(v_i), \mathrm{view}_{j+1}(v_i)) \in E, \\ 0, \text{ else.} \end{cases}$$

Note that $\mathrm{view}_1(v_i) = v_{i+1}$ and $\mathrm{view}_{|\mathrm{view}(v_i)|-1}(v_i) = v_{i-1}$ as every vertex sees its neighboring vertices on the polygon boundary. Therefore $\mathrm{cvv}_0(v_i) = \mathrm{cvv}_{|\mathrm{view}(v_i)|-1}(v_i) = 1$ for all $v_i \in V$.

Definition 4. *The* combinatorial visibility sequence cvs *of $\mathcal{P}$ lists all combinatorial visibility vectors of the individual vertices of $\mathcal{P}$ in ccw order:*

$$\mathrm{cvs} := (\mathrm{cvv}(v_0), \ldots, \mathrm{cvv}(v_{n-1})).$$

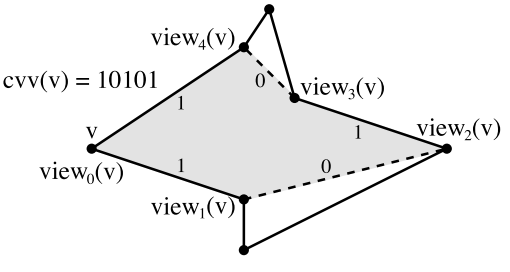


Fig. 1. Illustration of a combinatorial visibility vector

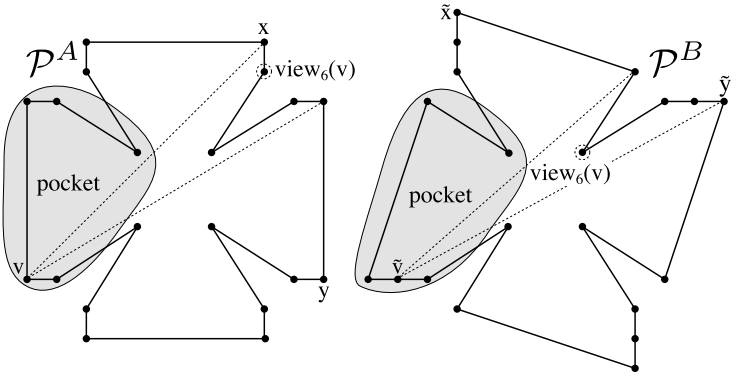


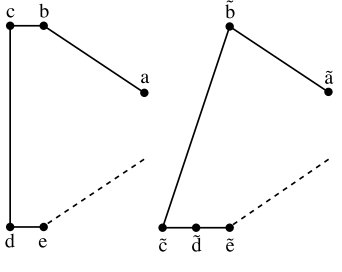
Fig. 2. Two polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ with identical cvs and different visibility

The following result implies that a robot constrained to moving along the boundary of the polygon and to sensing combinatorial visibilities can in general not reconstruct the visibility graph of a polygon.

Theorem 1. *The* cvs *of a polygon* $\mathcal{P}$ *does not uniquely define its visibility graph.*

Proof. In Fig. 2 we present two polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ which share the same cvs, yet have different visibility graphs. The proof is by inspection of the polygons together with the list of cvv's and view sequences of the relevant vertices in Fig. 3. Note that our construction is not in general position, however the polygons can easily be modified accordingly without changing visibilities or cvv's.

The idea behind the construction of the polygons is to use multiple copies of a "pocket" of vertices (cf. Fig. 2 for an illustration). Each pocket forms a convex curve, but the vertices connecting the pockets form reflex angles, resulting in a



vertex	cvv	view sequence
a $\tilde{a}$	1111101111011111	$abcdeadeabcabcde$ $\tilde{a}\tilde{b}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}\tilde{c}\tilde{d}\tilde{e}$
b $\tilde{b}$	11110111101	$bcdeadeabca$ $\tilde{b}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}$
c $\tilde{c}$	11101111011	$cdeadeabcab$ $\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}$
d $\tilde{d}$	11011110111	$deadeabcabc$ $\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}\tilde{c}$
e $\tilde{e}$	10111101111	$eadeabcabcd$ $\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}\tilde{c}\tilde{d}$

Fig. 3. The cvv and view sequence of every vertex within a pocket of $\mathcal{P}^A$ and $\mathcal{P}^B$, where a, b, c, d, e each refer to all four vertices at the corresponding position within their pocket in $\mathcal{P}_A$ and $\tilde{a}, \tilde{b}, \tilde{c}, \tilde{d}, \tilde{e}$ refer to their counterparts in $\mathcal{P}_B$

non-convex polygon $\mathcal{P}$. The vertices inside a pocket thus do not see all vertices of $\mathcal{P}$, they see (apart from their own pocket) only parts of exactly two pockets. We use the fact that the vertices have no way to distinguish what pockets they are "looking into" and we modify the polygon $\mathcal{P}^A$ by shifting the vertex c (cf. Fig. 2) so that in $\mathcal{P}^B$ the shifted vertex $\tilde{c}$ looks into different pockets, while not changing the cvv of any vertex. □

The polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ have twenty vertices each, however we were also able to construct similar polygons for $n = 12$. We were able to show that no examples exist for $n \leq 10$. We do not expect there to be examples with $n < 12$, we have however not been able to prove this. We did not present the polygons with twelve vertices as their construction is more involved.

The following theorem considers a related question about polygons with periodical cvs. We start by defining periodicity formally:

Definition 5. *We say that a* cvs $C = (C_0, C_1, \ldots, C_{n-1})$ *with* $C_i = \mathrm{cvv}(v_i)$ *is* periodical *with period* $p \geq 2$*, if* $C_i = C_{i+k\cdot\frac{n}{p}}$ *for all* $0 \leq i < n$ *and all* $1 \leq k < p$*. For each* $0 \leq i < n$ *we say* $\{v_{i+k\cdot\frac{n}{p}} | 0 \leq k < p\}$ *are* periodical partners.

The question is whether two vertices visible from periodical partners at the same local position in a polygon with periodical cvs have to be periodical partners themselves. A positive answer to this question would have an impact on various interesting problems in the field of simple robots; for example, on a weak version of the *rendezvous* problem in symmetrical polygons in which two identical, deterministic robots try to gain sight of each other. We show that it is not the case; we even show a stronger result.

Theorem 2. *There is a polygon* $\mathcal{P}$ *with a periodical* cvs *of period* $p \geq 2$ *for which we have*

$$\exists v_i \in V\, \exists j \in \{1, \ldots, |\mathrm{view}(v_i)| - 1\} : \mathrm{cvv}(\mathrm{view}_j(v_i)) \neq \mathrm{cvv}\left(\mathrm{view}_j\left(v_{i+\frac{n}{p}}\right)\right).$$

Proof. We construct a polygon $\mathcal{P}$ with period $p = 2$ with the aforementioned property from the two polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ in Fig. 2. The construction can easily be generalized to $p > 2$.

The idea of the construction is to "glue" $\mathcal{P}^A$ and $\mathcal{P}^B$ together at vertices v and $\tilde{v}$ of $\mathcal{P}^A$ and $\mathcal{P}^B$, respectively, where v and $\tilde{v}$ are as depicted in Fig. 2. We want to glue the polygons such that every two corresponding vertices w and $\tilde{w}$ of the two polygons form periodical partners in $\mathcal{P}$. Thus, we need to glue the polygons such that the cvv's of corresponding vertices w and $\tilde{w}$ are the same. We can then use the result of Theorem 1 which guarantees the existence of vertices w and $\tilde{w}$ with the same cvv but different views. Formally, if w from $\mathcal{P}^A$ is a vertex v_i in $\mathcal{P}$ and $\tilde{w}$ from $\mathcal{P}^B$ is a vertex $v_{i+n/2}$ in $\mathcal{P}$ (where n is the number of vertices of $\mathcal{P}$), there is a position j in their views such that $\mathrm{view}_j(v_i) = v_k$ and $\mathrm{view}_j\left(v_{i+\frac{n}{2}}\right) = v_l \neq v_{k+\frac{n}{2}}$. Because of the structure of the two polygons, we will have $\mathrm{cvv}(v_k) \neq \mathrm{cvv}(v_l)$ which proves the theorem.

The problem when gluing at $v/\tilde{v}$ is that these vertices have to be split in the process, which makes them distinguishable from all other vertices. By inserting

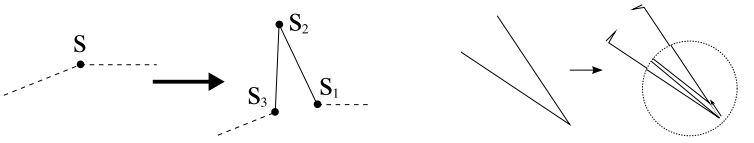


Fig. 4. Left: The concept of inserting spikes at vertices. Right: Illustration of how the spikes are inserted at reflex vertices. We chose our modification such that the right neighbor of the spike tip retains the visibility of the original vertex.

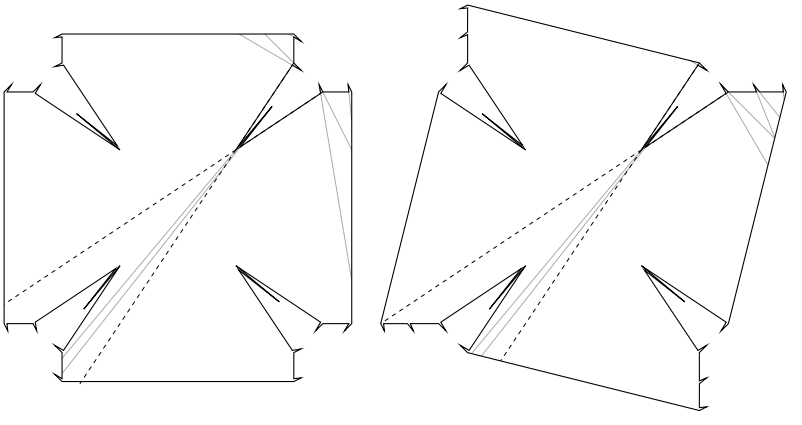
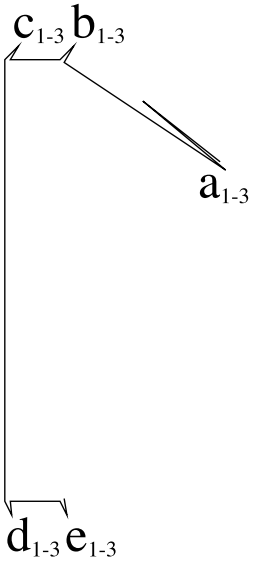


Fig. 5. The two polygons from Fig. 2 equipped with spikes and still with identical cvs. The areas visible from the different spike-tips are indicated.

spikes (cf. Fig. 4) at all vertices, we can again make vertices indistinguishable while still maintaining equal cvs'. Spikes can easily be inserted at convex vertices such that no distant vertex is visible from the spike tip and the spike tip's neighbors retain the vision of the original vertex (except for seeing vertices as gaps and seeing the spike tip). It is however not generally clear how to do that for reflex vertices, Fig. 4 shows how this can be done with the four reflex vertices in our case. Fig. 5 shows the spiked versions of $\mathcal{P}^A$ and $\mathcal{P}^B$ before gluing. Fig. 6 lists how the cvv's change with the introduction of spikes.

Once we have spiked versions of $\mathcal{P}^A$ and $\mathcal{P}^B$, we can glue them together in a straightforward way by simply splitting the spike tip of v and $\tilde{v}$ and attaching the open ends. It can easily be seen that the gluing does not break the periodicity of the cvs of $\mathcal{P}$. Fig. 7 shows the resulting polygon $\mathcal{P}$. The extension to $p > 2$ is easily made, as we can attach more than two copies of the two spiked polygons around a common center. □

Theorem 1 shows that the knowledge of the cvs is not sufficient to reconstruct the visibility graph of a polygon. A natural question is how to extend this information "minimally" in order to make the reconstruction possible. In the following we



vertex	cvv
a_1	1100101010100101010101
a_2	101
a_3	10101010100010101010010101010111
b_1	1110101010001010101001
b_2	101
b_3	101010100010101010100111
c_1	111010100010101010100101
c_2	101
c_3	10101000101010100010111
d_1	11101000101010100010101
d_2	101
d_3	10100010101010010010111
e_1	11100010101010010010101
e_2	101
e_3	10001010101001010101111

Fig. 6. The combinatorial visibilities of each vertex in a pocket of $\mathcal{P}^A$ after adding spikes (the same cvv's arise for $\mathcal{P}^B$). We write v_{1-3} to denote the group of vertices v_1, v_2, v_3.

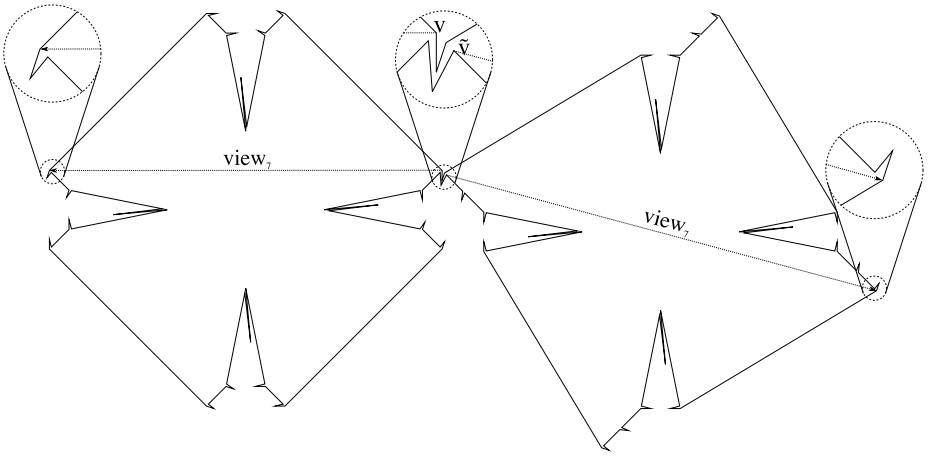


Fig. 7. The polygon with $n = 120$ that proves Theorem 2

show that adding the knowledge of all interior angles of the polygon is still not enough. We prove the following theorem.

Theorem 3. *The* cvs *and all interior angles of a polygon* $\mathcal{P}$ *do not uniquely determine the visibility graph of* $\mathcal{P}$*.*

Proof. Figure 8 shows a modified version of the polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ of Fig. 2. As one can easily check, the polygons still have the same cvs and different visibility graphs. In addition, they also have the same set of inner angles at the vertices. The existence of such polygons proves the theorem. □

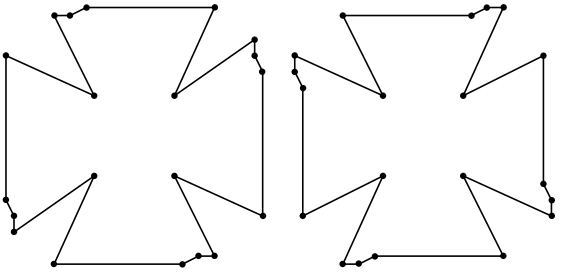


Fig. 8. Two polygons with identical cvs and identical interior angles but different visibility graphs. The visibilities are similar to those of $\mathcal{P}^A$ and $\mathcal{P}^B$.

Note that Theorems 1 and 3 do not imply that a robot equipped with sensors for measuring cvv's and/or inner angles cannot reconstruct the visibility graph, as such a robot would be able to distinguish $\mathcal{P}^A$ and $\mathcal{P}^B$ by moving to a vertex of the most distant corner and inspecting its cvv. However it seems difficult for such a robot to reconstruct the visibility graph - to prove that this indeed is not possible remains open.

4 Geometrical Sensors

In the previous section we saw that the simple combinatorial information we used is not enough to infer global properties of a polygon $\mathcal{P}$, namely its visibility graph. We now focus on geometrical sensors for measuring angles between distant vertices and show two sets of capabilities that enable a robot to reconstruct the visibility graph. We start by defining the two notions of angle sensors we consider.

Definition 6. *Let* $v_r \in V$ *be the vertex of polygon* $\mathcal{P}$ *which the robot is located at. We write* $\text{angle}(i,j)$ *with* $i < j$ *for the angle between the lines* $\overline{v_r \text{view}_i}$ *and* $\overline{v_r \text{view}_j}$ *in ccw direction. A sensor capable of determining* $\text{angle}(i,j)$ *for all* i,j *is called* angle sensor. *The type of the angle* $\text{angle}(i,j)$ *('reflex' or 'convex' depending on whether the angle is larger than* π *or not) is denoted by* $\text{angle\_type}(i,j)$*. A sensor capable of determining* $\text{angle\_type}(i,j)$ *for all* i,j *is called* angle-type sensor.

While it is clear that the angle sensor is stronger than the angle-type sensor, the angle-type sensor has the advantage that it is very robust with respect to measurement imprecision.

We have preliminary results that suggest that an angle sensor alone suffices for the reconstruction of the visibility graph. It is however not clear whether the angle-type sensor alone is sufficient, even when combined with the combinatorial sensor of Section 3. For a robot equipped with an angle-type sensor that is allowed to look back (cf. Section 2) however, we are able to prove the following result:

Theorem 4. *A robot with an angle-type sensor and with the ability to look back can uniquely reconstruct the visibility graph of any polygon $\mathcal{P}$.*

Proof. We prove this by presenting an algorithm for the robot to construct the visibility graph.

The robot moves from vertex to vertex along the boundary of $\mathcal{P}$ in ccw order. At each vertex v_i it iteratively identifies all visible vertices. It starts by identifying the vertices view_1, $\text{view}_{|\text{view}|-1}$ which trivially have the global index $i+1, i-1$. Further vertices can be identified as follows:

Let v_k be the first visible vertex in ccw order that has not yet been identified and v_j be the previous vertex that is visible, so that j is known to the robot and it needs to find k. The robot does this by counting all vertices "beyond" v_j and those "beyond" v_k. In order to understand the notion of vertices lying beyond some vertex v_b, consider the intersection x of the ray from v_i to v_b with the boundary of $\mathcal{P}$. The vertices between (either in ccw order or in clockwise order) x and v_b are said to lie beyond v_b. The total number of the vertices beyond v_j and v_k (in ccw order and clockwise order, respectively) then simply needs to be added to $j+1$ in order to obtain k (cf. Fig. 9).

It remains to be seen how the robot situated at v counts the number of vertices beyond another vertex with local index b. The first step is moving to b. By looking back, the robot can identify v in its new view. Therefore it can decide which of the now visible vertices form a reflex angle with v and are thus

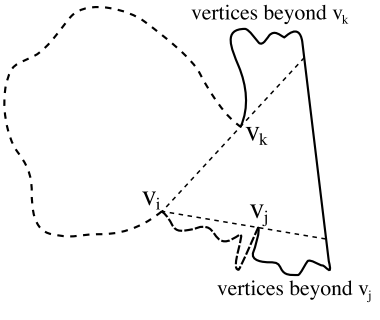


Fig. 9. Visualization of the procedure for inferring the global index of v_k if the previous vertex v_j has already been identified. It is enough to count the number of vertices beyond v_j and v_k as those are the ones between v_j and v_k in ccw order.

behind the current vertex b when looking from v. All these vertices have to be counted as well as the number of vertices beyond them (which in turn are not visible to b). This is done recursively, so that at the end the robot sums up all vertices that are directly or indirectly behind b. The following listing shows the procedure for counting the vertices behind b in pseudocode.

function beyond(b,o)
input: local index b, order o that specifies on which side of vision$_b$ to count
output: *count* of vertices beyond vision$_b$ w.r.t. the current position of the robot

1. $count \leftarrow 0$
2. **move to** b
3. $i \leftarrow$ **look back**
4. **for each** $j \in [1, \ldots, |\text{view}| - 1]$ with $(o = \mathbf{ccw} \wedge j < i) \vee (o = \mathbf{cw} \wedge j > i)$ **do**
5. **if** $\text{type}(j, i) = \mathbf{reflex}$
6. $count \leftarrow count + 1 + \text{beyond}(j, \mathbf{ccw}) + \text{beyond}(j, \mathbf{cw})$
7. **move to** i

In order to prove the correctness of our algorithm, we need to show that no vertex is counted twice when counting the vertices beyond v_j and beyond v_k. The two calls of beyond() for v_j and v_k consider distinct sets of vertices as the first considers those to the right of the line to v_j and the second considers those on the left of the line to v_k. As v_k by definition lies left of v_j, there is no overlap. We show that a single call to beyond() does not count vertices twice either: It is obvious that the recursive calls in line 6 consider distinct sets of vertices, as one considers only vertices on the left and the other only on the right of view$_j$. The only possible overlap could be between two calls of the form beyond(x, **ccw**), beyond(y, **cw**) with $x < y$. Again, because by definition view$_y$ lies to the right of view$_x$, there can be no overlap. The entire algorithm is at no point ambiguous, so that the solution found has to be unique. □

We can enable a robot with angle sensor to emulate the robot from Theorem 4 by giving it a *compass*. A compass provides the robot with a global reference direction. The angle sensor combined with a compass can measure the global direction to each vertex in sight.

Definition 7. *Let $p = (0, \infty)$. Let v_r be the position of the robot and* view$'$ *be the view of the robot if p was a vertex of $\mathcal{P}$ visible to v_r. A* compass *enables a robot to determine the index i for which $p =$ view$'_i$. When combined with an angle sensor, a compass also provides the angles between the lines $\overline{v_r\text{view}'_i}$ and $\overline{v_r\text{view}'_j}$ in ccw direction, for all indices j.*

The next theorem follows immediately from Theorem 4.

Theorem 5. *A robot with an angle sensor and a compass can uniquely reconstruct the visibility graph of any polygon $\mathcal{P}$.*

Proof. The angle sensor can obviously emulate an angle type sensor. It therefore suffices to show that the robot can imitate the capability of looking back and

thus apply the strategy described in the proof of Theorem 4. Assume the robot moves from a vertex v to a vertex u that it sees in the global direction d. From its new location u the robot knows that v lies in direction $-d$. Because of general position, the robot is guaranteed to see only v in that direction and thus the robot is capable of uniquely identifying the vertex it came from, in other words the robot is capable of looking back. □

Note that the last two results do not rely on the knowledge of n and that in fact the corresponding robots are capable of inferring n.

5 Conclusion

We have studied the problem of reconstructing the visibility graph of a polygon $\mathcal{P}$ using simple robots. In this context, we have discussed three different configurations of sensors for simple robots. We have proven that the two configurations based on geometrical sensor enable the robot to infer the visibility graph of a polygon while purely combinatorial knowledge, in terms of the cvs of the polygon, does not suffice. In addition we have shown a property of symmetric polygons which makes combinatorial visibility even weaker in that case.

It is clear that a robot with one of the two geometrical sensor configurations is stronger than a robot equipped with the combinatorial sensor, as combinatorial visibilities can be derived from the visibility graph. The task of finding the weakest configuration that allows reconstructing the visibility graph remains unsolved.

References

1. Dudek, G., Freedman, P., Hadjres, S.: Mapping in unknown graph-like worlds. Journal of Robotic Systems 13(8), 539–559 (1998)
2. Dudek, G., Jenkins, M., Milios, E., Wilkes, D.: Robotic exploration as graph construction. IEEE Transactions on Robotics and Automation 7(6), 859–865 (1991)
3. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In: Aggarwal, A.K., Pandu Rangan, C. (eds.) ISAAC 1999. LNCS, vol. 1741, pp. 93–102. Springer, Heidelberg (1999)
4. Ganguli, A., Cortés, J., Bullo, F.: Distributed deployment of asynchronous guards in art galleries. In: Proceedings of the American Control Conference, pp. 1416–1421 (2006)
5. Ghosh, S.K.: Visibility Algorithms in the Plane, 1st edn. Cambridge University Press, Cambridge (2007)
6. Hoffmann, F., Icking, C., Klein, R., Kriegel, K.: The polygon exploration problem. SIAM Journal on Computing 31(2), 577–600 (2001)
7. O'Kane, J.M., LaValle, S.: Localization with limited sensing. IEEE Transactions on Robotics 23(4), 704–716 (2007)
8. LaValle, S.: Planning Algorithms. Cambridge University Press, Cambridge (2006)

9. Oommen, B., Iyengar, S., Rao, N., Kayshap, R.: Robot navigation in unknown terrains using learned visibility graphs. Part I: The Disjoint Convex Obstacle Case. IEEE Journal of Robotics and Automation RA-3(6), 672–681 (1987)
10. Suri, S., Vicari, E., Widmayer, P.: Simple robots with minimal sensing: From local visibility to global geometry. International Journal of Robotics Research 27(9), 1055–1067 (2008)
11. Yershova, A., Tovar, B., Ghrist, R., LaValle, S.: Bitbots: Simple robots solving complex tasks. In: Proceedings of the Twentieth National Conference on Artificial Intelligence, pp. 1336–1341 (2005)

Stability of Networks in Stretchable Graphs*

Davide Bilò[1], Michael Gatto[1], Luciano Gualà[2],
Guido Proietti[3,4], and Peter Widmayer[1]

[1] Institut für Theoretische Informatik, ETH Zurich, 8092 Zürich, Switzerland
[2] Dipartimento di Matematica, Università di Tor Vergata, 00133 Roma, Italy
[3] Dipartimento di Informatica, Università di L'Aquila, 67010 L'Aquila, Italy
[4] Istituto di Analisi dei Sistemi ed Informatica, CNR, 00185 Roma, Italy
proietti@di.univaq.it, guala@mat.uniroma2.it,
{dbilo,gattom,widmayer}@inf.ethz.ch.

Abstract. In classic optimization theory, the concept of *stability* refers to the study of how much and in which way the optimal solutions of a given minimization problem Π can vary as a function of small *perturbations* of the input data. Motivated by congestion problems arising in shortest-path based communication networks, in this paper we restrict ourselves to the case in which Π is actually a network design problem on a given graph $G = (V, E, w)$ of $|V| = n$ nodes, $|E| = m$ edges, and with a positive real weight $w(e)$ on each edge $e \in E$. We focus on a subclass of perturbations, that we call *stretching perturbations*, in which the weights of the edges of G can be increased by at most a fixed multiplicative real factor $\lambda \geq 1$.

For this class of perturbations, we address the problem of computing the *stability number* of any given subgraph H of G containing at least an optimal solution of Π, namely the maximum stretching factor for which H keeps on maintaining an optimal solution. Furthermore, given a stretching factor λ, we study the problem of constructing a *minimal* subgraph of G with stability number greater or equal to λ.

We develop a general technique to solve both problems. By applying this technique to the *minimum spanning tree* and the *single-source shortest paths tree (SPT)* problems, we obtain $\mathcal{O}(m\alpha(m, n))$ and $\mathcal{O}(mn(m + n \log n))$ time algorithms, respectively, where $\alpha(\cdot, \cdot)$ is the functional inverse of Ackermann's function. Furthermore, for the SPT problem, we show that if H coincides with the set of *all* optimal solutions, then the time complexity can be reduced to $\mathcal{O}(mn)$. Finally, for the *single-source single-destination shortest path* problem, if the optimal solutions of the input instance happen to form a set of vertex-disjoint paths, and H coincides with this set, then we show that we can compute the stability number in $\mathcal{O}(mn + n^2 \log n)$ time.

Keywords: Communication Networks, Shortest Paths, Edge Perturbation, Stability Theory.

* Part of this work has been developed while the fourth author was visiting ETH Zurich.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 100–112, 2010.

1 Introduction

Let a communication network be modeled by a (either directed or undirected) graph $G = (V, E, w)$ of $|V| = n$ nodes, $|E| = m$ edges, and with a positive real weight $w(e)$ on each edge $e \in E$. A *network communication problem* Π asks for computing a subgraph S of G such that (i) S belongs to a set $\mathcal{F}$ of feasible solutions, and (ii) S minimizes an objective function $\mu(S, w)$ which depends on the weights of the edges of S.[1] However, in many practical situations, edge weights in G may be susceptible to sudden changes, due to unpredictable boundary conditions, and thus the question of assessing the quality of a selected solution in the presence of such volatility arises naturally.

This is especially true in the domain of information routing, where link congestion phenomena are frequent, for that the performances of a given adopted solution might degrade rapidly. Since pursuing the efficiency of the communication system is ineludible, in an ideal situation one should be able to cope with these undesired events by constantly using an optimal route. However, this can be done only by spending a corresponding effort in terms of both computational costs (to maintain dynamically an optimal solution), and execution of rerouting operations. Therefore, to mitigate this tension between the search of optimality and the costs to actually implement it, one should try to improve his overall knowledge about the network itself. For instance, to avoid unnecessary computations, it would be helpful to know in advance what the affordable congestion threshold of a given set of used links is, namely how much these links can be congested until the optimality of the corresponding route is affected. Orthogonally, to avoid rerouting, one might decide to be resilient to congestion phenomena *up to* a prefixed threshold, by making use of a *redundant routing network* in which a set of additional routes is superimposed to an optimal route, so that optimal performances are guaranteed in spite of arbitrary link congestions within the selected threshold. Notice that such a network might also be useful in a perspective of reducing computational costs, since as long as link congestion does not go over the prefixed threshold, it enables to perform the dynamic maintenance of the optimal solution on a (small) subgraph of the underlying communication graph. Summarizing, to appropriately tolerate congestions, it is crucial to establish a good trade-off between the efficiency and the robustness of the system.

Problem definition and our results. Along this line of research, in this paper we consider a scenario in which multiple link congestions at a time may occur, though not in an arbitrary way. More precisely, given a network communication problem Π on G, we focus on a subclass of perturbations, that we call *stretching perturbations*, in which edge weights in G can be amplified at most by a fixed multiplicative real factor $\lambda \geq 1$. This setting is motivated from the observation that link congestions can be physically described by means of a linear dilatation of the corresponding edge weights, i.e., a stretching perturbation.

[1] We tacitly assume that the set $\mathcal{F}$ does not depend on the weight function $w(\cdot)$, and that $\mu(S, w)$ strictly increases as any edge of S increases its weight.

More formally, given a real value $\lambda \geq 1$, a vector $\bar{\lambda} = \langle \bar{\lambda}_{e_1}, \ldots, \bar{\lambda}_{e_m} \rangle \in \mathbb{R}^m$ is called a λ-*perturbation* if $1 \leq \bar{\lambda}_{e_j} \leq \lambda$ for every e_j. For any such perturbation, we denote by $G_{\bar{\lambda}} = (V, E, w_{\bar{\lambda}})$ the graph obtained from G by amplifying the weight of each edge e_j by a multiplicative factor $\bar{\lambda}_{e_j}$, i.e., $w_{\bar{\lambda}}(e_j) = \bar{\lambda}_{e_j} w(e_j)$ for each $e_j \in E$. Then, given a subgraph H of G, we define the *stability number of* H *w.r.t.* Π as the maximum value $\delta(H)$ such that, for every $\delta(H)$-perturbation $\bar{\lambda}$, H contains an optimal solution for $G_{\bar{\lambda}}$ w.r.t. Π. Clearly, $\delta(G) = +\infty$. Moreover, we assume $\delta(H) = -\infty$, when H contains no optimal solution. A simple example of the notion of stability number is shown in Figure 1. To assess the effectiveness of a selected solution under this class of perturbations, we specifically aim to address the following two interlaced problems:

($\mathbf{Q}_1$) Given a subgraph H of G, compute $\delta(H)$.
($\mathbf{Q}_2$) Given a real value $\lambda \geq 1$, find a minimal subgraph H of G with $\delta(H) \geq \lambda$.

Besides, we will also consider the following subproblem of Q_1, due to its immediate practical relevance:

($\mathbf{Q}_1^*$) Compute $\delta(H^*)$, where H^* is the subgraph of G made up by the union of all the optimal solutions for G w.r.t. Π.

In the paper, we develop a general technique to solve the aforementioned problems. By applying this technique to the classic *minimum spanning tree (MST)* problem, we obtain an $\mathcal{O}(m\alpha(m,n))$ time algorithm solving all the considered problems, where $\alpha(\cdot,\cdot)$ is the functional inverse of Ackermann's function [8]. On the other hand, for the other classic *single-source shortest paths tree (SPT)* problem, we show that Q_1 and Q_2 can be solved in $\mathcal{O}(mn(m + n\log n))$ time, while Q_1^* can be solved in $\mathcal{O}(mn)$ time. Finally, for the *single-source single-destination shortest path (SP)* problem, if the optimal solutions of the input instance happen to form a set of vertex-disjoint paths, then we show that Q_1^* can be solved in $\mathcal{O}(mn + n^2 \log n)$ time.

Related work. In their essence, our congestion tolerance problems can be regarded as *stability problems*. Indeed, stability theory studies exactly how much and in which way the solutions of a given optimization problem can vary as a function of small *perturbations* of the input data. From an algorithmic

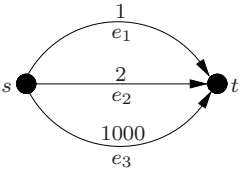


Fig. 1. A simple graph in which we want to route messages from s to t via a shortest path. The stability number of the subgraph consisting of edge e_1 is 2. Augmenting such subgraph with the edge e_2 let the stability number increase to 1000.

perspective, two main methods have been developed in the literature to deal with uncertainty: *robust optimization* and *sensitivity analysis*.

In robust optimization, data uncertainty is expressed through a specification of a feasible set of values for each of the input data, over which one tries to find a *compromise solution* hedging against the worst instances that might arise. The compromise should obey to a given *robustness criterium*, which in its turn depends on the problem specification. In their comprehensive work [4], Kouvelis and Yu defined several robustness criteria, and we refer the reader to [1] for a recent survey on the topic. Closer to the spirit of our work is instead the other way of dealing with uncertainty, namely that of performing the so-called *sensitivity analysis* of an optimal solution. This is a post-optimality study initially defined in [7], which analyzes how long the solution stays optimal in spite of *single* changes in the input data. To illustrate it more concretely, let us focus our attention again on the MST problem. Here the problem is to establish how much the weight of each individual edge in the MST can be perturbed before the spanning tree is no longer minimal. In his seminal paper [8], Tarjan considered the case of a general graph G with n nodes and m edges, and solved the problem in $\mathcal{O}(m\,\alpha(m,n))$ time; this result was then improved to $\mathcal{O}(m \log \alpha(m,n))$ [6]. Tarjan also considered the other fundamental SPT problem, for which he similarly provided an $\mathcal{O}(m\,\alpha(m,n))$ time sensitivity analysis. Later on, sensitivity analysis has been applied to many other network optimization problems, and we refer the reader to [3] for extensive references.

Paper organization. The paper is structured as follows. In Section 2, we explain the general technique to solve the defined stability problems. We apply this technique to the MST Problem in Section 3. Section 4 deals with the SPT stability problem, while the SP problem is considered in Section 5.

2 A General Technique

In this section we introduce the concept of optimality thresholds of the edges and we show how these values are related to the stability number of a given subgraph. Let $G = (V, E, w)$ be an edge-weighted either directed or undirected graph with $|V| = n$, and $|E| = m$ (we assume that $w(e) > 0$ for every edge $e \in E$). In the following, given any $e \in E, v \in V$, we use $G - e$ and $G - v$ to denote the graph obtained from G by discarding e and v (with its incident edges), respectively. Given $U \subseteq V$, $G - U$ is defined in a similar way. Finally, for any subgraph H and edge e, $H + e$ stands for $H = (V(H), E(H) \cup \{e\})$.

Definition 1. *The* optimality threshold $\gamma(e)$ *of* $e \in E$ *is defined as the minimum real value* $\lambda \geq 1$ *(if any) for which there exists a feasible solution* $S \in \mathcal{F}$ *such that (i)* S *is an optimal solution for some* λ*-perturbation of* G*, and (ii)* $e \in E(S)$*. If such a value does not exist, then* $\gamma(e) = \infty$*.*

Lemma 1. *Let* H *be a proper subgraph of* G *containing at least an optimal solution, then* $\delta(H) = \min_{e \notin E(H)} \gamma(e)$*.*

Proof. Let $\eta = \min_{e \notin E(H)} \gamma(e)$, and let $\tilde{e} \in E \setminus E(H)$ be an edge such that $\gamma(\tilde{e}) = \eta$. The proof is by contradiction. Assume $\eta > \delta(H)$. Then, let us consider a value η' such that $\delta(H) < \eta' < \eta$. By definition of $\delta(H)$, there must exist an η'-perturbation, say $\bar{\lambda}$ such that H contains no optimal solution for $G_{\bar{\lambda}}$. Then, let S be such a solution. It is clear that S contains an edge, say e', which does not belong to H. This implies that $\gamma(e') \leq \eta' < \eta$. This is a contradiction.

On the other hand, assume $\eta < \delta(H)$. By definition of η, there exists an η-perturbation, say $\bar{\lambda}'$, and a feasible solution $\tilde{S}$ such that $\tilde{e} \in E(\tilde{S})$, and $\tilde{S}$ is optimal in $G_{\bar{\lambda}'}$. Now, consider the perturbation $\bar{\lambda}''$ defined as follows: $\bar{\lambda}''_e = \bar{\lambda}'_e$ if $e \in \tilde{S}$, $\delta(H)$ otherwise. Then, for each feasible solution S' contained in H we have:

$$\mu(\tilde{S}, w_{\bar{\lambda}''}) = \mu(\tilde{S}, w_{\bar{\lambda}'}) \leq \mu(S', w_{\bar{\lambda}'}) < \mu(S', w_{\bar{\lambda}''}),$$

where the last inequality holds since $\eta < \delta(H)$. Hence, $\tilde{S}$ is a feasible solution that is strictly better for $G_{\bar{\lambda}''}$ than every solution contained in H, which means that H cannot contain an optimal solution for $G_{\bar{\lambda}''}$. This is a contradiction since $\bar{\lambda}''$ is a $\delta(H)$-perturbation. □

As a consequence of the above lemma, we have:

Corollary 1. *There exists a polynomial-time algorithm for computing $\gamma(e)$ for every $e \in E$ if and only if Q_1 can be solved in polynomial time.*

Proof. One direction follows immediately from Lemma 1. For the other direction is concerned, assume that we can solve Q_1 in polynomial time. Then, by using Lemma 1, it is easy to see that for every edge $e \in E$, $\gamma(e) = \delta(G - e)$ if $\delta(G - e) \neq -\infty$, 1 otherwise. □

Corollary 2. *Given $\lambda > 1$, the minimal subgraph H of G with $\delta(H) \geq \lambda$ is $H = (V, E_\lambda)$, where $E_\lambda = \{e \in E \mid \gamma(e) < \lambda\}$. In other words, if there exists a polynomial-time algorithm for computing $\gamma(e)$ for every $e \in E$, then Q_2 can be solved in polynomial time.* □

3 The Minimum Spanning Tree Stability Problem

As a simple application of the previous results, let us consider the *minimum spanning tree (MST)* problem which, given a weighted undirected graph $G = (V, E, w)$, asks for computing a spanning tree T of G such that the cost of T, i.e. $\sum_{e \in E(T)} w(e)$, is minimum. We have the following:

Theorem 1. *For the MST problem, Q_1, Q_2 and Q_1^* can be solved in $\mathcal{O}(m\,\alpha(m, n))$ time.*

Proof. It suffices to show how to compute the optimality thresholds of all edges in $\mathcal{O}(m\,\alpha(m, n))$ time. We first compute an MST T of G. This can be done in $\mathcal{O}(m\,\alpha(m, n))$ time [5]. It is clear that $\gamma(e) = 1$ for every $e \in E(T)$. Moreover, observe that for each $f = (x, y) \in E \setminus E(T)$, we have that $\gamma(f) = w(f)/w(e_f)$,

where e_f is an edge of maximum weight among the ones belonging to the (unique) path in T joining x and y. The problem of computing e_f for each non-tree edge f is a well-known problem called the *MST verification* problem, which can be solved in $\mathcal{O}(m)$ time [2]. □

4 The Single-Source Shortest Path Tree Stability Problem

Let $G = (V, E, w)$ be a directed graph with n vertices and m edges having a positive weight $w(e)$ associated to each edge $e \in E$. Let $s, t \in V$ be two vertices of G. A *shortest path P from s to t* is an $s-t$ (simple) path of minimum length, where the *length of a path* is defined to be the sum of its edge weights. We denote by $d_G(s,t)$ the *distance* from s to t in G, i.e., the total length of any shortest $s-t$ path in G (if no $s-t$ path exists, then $d_G(s,t) = \infty$). Given a path P and two vertices $u, v \in V(P)$, we denote by $P[u,v]$ the $u-v$ subpath of P, and, for any perturbation vector $\bar{\lambda}$, we denote by $w(P, \bar{\lambda})$ the length of P in $G_{\bar{\lambda}}$, i.e. $w(P, \bar{\lambda}) = \sum_{e \in E(P)} \bar{\lambda}_e w(e)$. The *single-source shortest paths tree* problem (SPT) asks for a directed tree T rooted at a given source vertex $s \in V$ that minimizes the overall sum of distances from the source to every vertex in the graph, i.e., the value $\sum_{v \in V} d_T(s,v)$.

This section begins with a description of a polynomial time algorithm that computes the value $\gamma(e)$ in (G, s) w.r.t. the SPT problem, where edge $e = (x, y) \in E$ is given as input, and then it describes a faster algorithm to solve Q_1^*.

4.1 The Algorithm for Computing the Optimality Thresholds

The algorithm keeps track of an upper bound of $\gamma(e)$ in the variable `ub` and refines this value through iterative operations until it becomes exactly equal to $\gamma(e)$. During iteration $i \geq 1$, the algorithm computes the graph H_i of all the shortest s-x paths in G', where $G' = G$ at the beginning of iteration 1. Graph H_i is needed to associate the value $\texttt{cost}_i(v) = \frac{d_{H_i}(v,x)+w(e)}{d_G(v,y)}$ to every vertex $v \in V(H_i)$. The value $\texttt{cost}_i(v)$ is a lower bound of the least value λ for which there exists a λ-perturbation $\bar{\lambda}$ such that $G_{\bar{\lambda}}$ contains a shortest s-y path passing through both v and e. Then, the algorithm computes an s-x path P_i in H_i that minimizes the *bottleneck w.r.t. function* $\texttt{cost}_i$, i.e., the value $B_i = \max_{v \in V(P_i)} \texttt{cost}_i(v)$. The algorithm concludes iteration i by updating the value of `ub` with B_i, in case $B_i < \texttt{ub}$, and by deleting from G' all vertices whose corresponding $\texttt{cost}_i$ values are greater or equal to `ub`. The algorithm stops iterating when no s-x path in G' is left. For a more formal description of the algorithm see Algorithm 1, while an example of its execution is given in Figure 2.

Let $G = (V, E, w)$ be a graph and let s and t be two distinct vertices of G. We say that $U \subseteq V$ is an *s-t cut of G* if $s, t \notin U$ and no s-t path exists in the graph $G - U$. We are now ready to prove the following

Proposition 1. *At the end of iteration i an s-x cut of H_i has been removed from G'.*

Algorithm 1

Require: An edge-weighted graph $G = (V, E, w)$, weight function $w : E \to \mathbb{R}^+$, a source $s \in V$, and an edge $e = (x, y) \in E$.
Ensure: The optimality threshold $\gamma(e)$ in G w.r.t. SPT problem with source s.

1: $\mathtt{ub} = +\infty$
2: $G' = G$
3: $i = 0$
4: **while** there exists an s-x path in G' **do**
5: $\quad i = i + 1$
6: $\quad$ compute the structure H_i of all the s-x shortest paths in G'
7: $\quad$ **for all** $v \in V(H_i)$ **do**
8: $\quad\quad \mathtt{cost}_i(v) = \frac{d_{H_i}(v,x)+w(e)}{d_G(v,y)}$
9: $\quad$ **end for**
10: $\quad$ compute an s-x path P_i in H_i that minimizes the bottleneck w.r.t. $\mathtt{cost}_i$. Let $B_i = \max_{v \in V(P_i)} \mathtt{cost}_i(v)$
11: $\quad \mathtt{ub} = \min\{\mathtt{ub}, B_i\}$
12: $\quad$ **for all** $v \in V(H_i)$ **do**
13: $\quad\quad$ **if** $\mathtt{cost}_i(v) \geq \mathtt{ub}$ **then**
14: $\quad\quad\quad G' = G' - v$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end while**
18: **return** ub

Proof. Let P_i be the s-x path of H_i that is computed by the algorithm at iteration i and let $B_i = \max_{v \in V(P_i)} \mathtt{cost}_i(v)$. After the execution of Line 11 of the algorithm we have that $\mathtt{ub} \leq B_i$. As P_i is an s-x path in H_i that minimizes the bottleneck w.r.t. $\mathtt{cost}_i$, then every other s-x path in H_i has to contain a vertex whose corresponding $\mathtt{cost}_i$ value is greater or equal to B_i. Therefore, at least one vertex per s-x path in H_i has been removed from G' after the execution of the for-loop in Lines 12–16. Thus, the set of vertices removed from G' at the end of iteration i is an s-x cut of H_i. □

Lemma 2. *During the execution of Algorithm 1 we have* $\gamma(e) \leq \mathtt{ub}$.

Proof. Let us consider any iteration i of the algorithm. Let P_i be the path from s to x in H_i computed by the algorithm during iteration i and let B_i be its bottleneck value w.r.t. $\mathtt{cost}_i$. Let $\bar{\lambda}$ be the B_i-perturbation of G where all edges in $E \setminus E(P_i)$ are perturbed by B_i while edges of P_i are not perturbed. In order to prove the claim, first notice that it is enough to show that $G_{\bar{\lambda}}$ contains a shortest path from s to y passing through edge e. Let P be any shortest s-y path in $G_{\bar{\lambda}}$ and let $v \in V(P)$ be the farthest vertex from s that is also in P_i. Clearly, $P[v, y]$ is a shortest v-y path in $G_{\bar{\lambda}}$. Moreover, it is vertex disjoint w.r.t. $P_i[v, y]$. Thus, as $\mathtt{cost}_i(v) \leq B_i$ and because P_i is a shortest s-x path in H_i, we have that

$$w(P_i[v, y], \bar{\lambda}) = w(P_i[v, y]) = \mathtt{cost}_i(v) \cdot d_G(v, y) \leq B_i \cdot w(P[v, y], \bar{\lambda}).$$

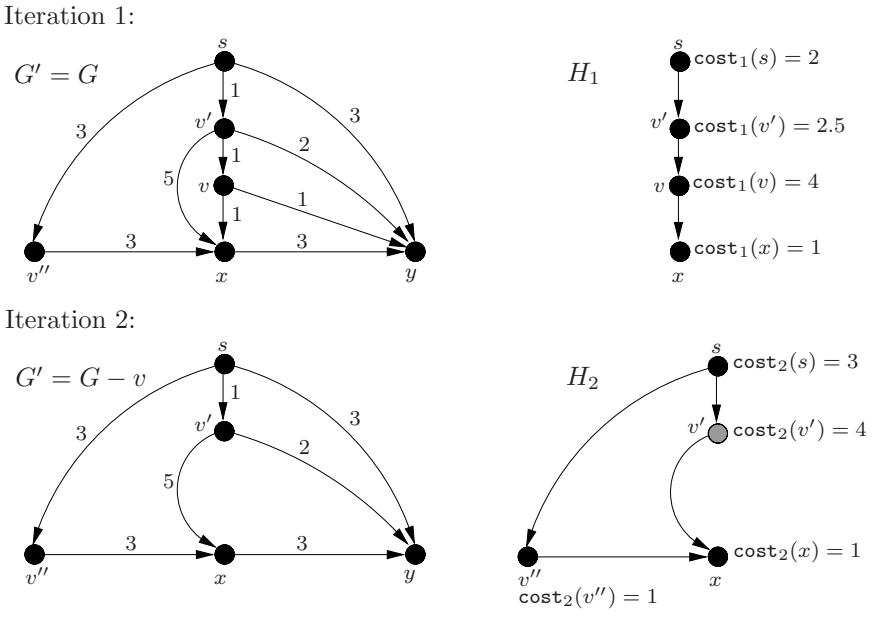


Fig. 2. An example of the execution of Algorithm 1 to compute the optimality threshold of edge (x, y). On the left side, it is shown the graph G' at the beginning of every iteration. On the right side, the graph H_i and the $\texttt{cost}_i$ values computed during iteration i are shown. Path P_i is given by the black vertices in H_i. At the end of iteration 1, $\texttt{ub} = 4$ and v has been removed from G'. At the end of iteration 2, $\texttt{ub} = 3$ and vertices s, v' have been removed from G'. The algorithm stops after two iterations as no s-x path in G' is left.

Therefore, $P[s, v] \cup P_i[v, y]$ is a shortest s-y path in $G_{\bar{\lambda}}$ that passes through edge e. The claim follows. □

Lemma 3. *Let P be a shortest s-y path passing through edge e in some $\gamma(e)$-perturbation of G. Then, $\gamma(e) \geq \max_{v \in V(P)} \frac{w(P[v,y])}{d_G(v,y)}$.*

Proof. Let $B = \max_{v \in V(P)} \frac{w(P[v,y])}{d_G(v,y)}$ and let v^* be a vertex of P such that $B = \frac{w(P[v^*,y])}{d_G(v^*,y)}$. Let P' be a shortest v^*-y path in G and let $F = E(P[v^*, y]) \cap E(P')$. W.l.o.g., we can assume that $P \neq P'$ and thus $w(F) < d_G(v^*, y)$. Let $\bar{\lambda}$ be the $\gamma(e)$-perturbation where all edges in $E \setminus E(P)$ are perturbed by $\gamma(e)$ while all other edges are not perturbed. As P is a shortest s-y path in some $\gamma(e)$-perturbation of G, it is a shortest s-y path in $G_{\bar{\lambda}}$. Therefore, $P[v^*, y]$ is a shortest v^*-y path in $G_{\bar{\lambda}}$. As a consequence,

$$w(P', \bar{\lambda}) = \gamma(e) \cdot d_G(v^*, y) - (\gamma(e) - 1)w(F) \geq w(P[v^*, y], \bar{\lambda}) = w(P[v^*, y])$$

implies

$$\gamma(e) \geq \frac{w(P[v^*, y]) - w(F)}{d_G(v^*, y) - w(F)}.$$

Now, using the fact that $0 \leq w(F) < d_G(v^*, y)$, we obtain $\gamma(e) \geq \frac{w(P[v^*,y])}{d_G(v^*,y)} = B$, thus proving the claim. □

Lemma 4. *Let P be a shortest s-y path passing through edge e in some $\gamma(e)$-perturbation of G. Let i be the iteration in which the first vertex in $V(P)$ is removed from G'. At the end of iteration i it is $\mathtt{ub} = \gamma(e)$.*

Proof. Let v be the first vertex in $V(P)$ that is removed from G' at iteration i. Clearly, $P[v, x]$ is a v-x path in the graph G' at the beginning of iteration i. As a consequence, we have that

$$\mathtt{cost}_i(v) = \frac{d_{H_i}(v,x) + w(e)}{d_G(v,y)} \leq \frac{w(P[v,x]) + w(e)}{d_G(v,y)} \overset{\text{Lm 3}}{\leq} \gamma(e) \overset{\text{Lm 2}}{\leq} \mathtt{ub}.$$

Since v is removed from G' because $\mathtt{cost}_i(v) \geq \mathtt{ub}$, it follows that all above inequalities are satisfied with equality. Hence, $\gamma(e) = \mathtt{ub}$. □

Theorem 2. *Algorithm 1 computes the value $\gamma(e)$ in $\mathcal{O}(n(m + n \log n))$-time.*

Proof. The correctness of the algorithm follows from Lemma 4. As far as the time complexity is concerned, observe that every iteration (while-loop condition included) takes $\mathcal{O}(m + n \log n)$ time. Moreover, Proposition 1 implies that at most n iterations are needed. □

The above theorem implies the following result.

Corollary 3. *For the single-source shortest paths tree problem both Q_1 and Q_2 can be solved in $\mathcal{O}(nm(m + n \log n))$-time.* □

4.2 A Faster Algorithm for Q_1^*

In the remaining of this section, we denote by H^* the subgraph of G made up of the union of all the SPT's of G rooted at s. In what follows, we develop an efficient $\mathcal{O}(nm)$-time algorithm to compute the value $\sigma := \delta(H^*)$. The algorithm makes use of the ideas of Algorithm 1 together with some additional observations. In order to be time efficient, the algorithm precomputes the following information:

1. the distances $d_G(s, v)$ from s to every other vertex $v \in V$;
2. the set $A_v = \{u \in V \mid \exists \text{ a } u\text{-}v \text{ path in } H^*\}$ for every vertex $v \in V$;
3. a value η_v such that $\eta_v = \min_{u \in V \mid (u,v) \in E(H^*)} d_G(s, u)$ for every vertex $v \in V$.

Then, for each edge $e = (x, y) \in E \setminus E(H^*)$, the algorithm computes the value $\eta^* = \min_{u \in A_x \setminus A_y} \eta_u$ and the value $\tau(e) = \frac{d_G(s,x) - \eta^* + w(e)}{d_G(s,y) - \eta^*}$. If $A_x \setminus A_y$ is empty, then $\eta^* = 0$. Finally, the algorithm computes the value $\min_{e \in E \setminus E(H^*)} \tau(e)$ and outputs it.

Theorem 3. *There exists an $\mathcal{O}(nm)$-time algorithm that computes the value σ.*

Proof. We start analyzing the time complexity of the above algorithm. From the description of the algorithm it is easy to see that each value $\tau(e)$ can be computed in $\mathcal{O}(n)$ time. Moreover, $\mathcal{O}(nm)$-time is sufficient for all the information the algorithm precomputes.

As long as correctness of the algorithm is concerned, we begin our proof with some crucial observations. Let H' be a subgraph of G containing some SPT of G rooted at s. Let $\bar{\lambda}$ be a $\delta(H')$-perturbation of G. Observe that $G_{\bar{\lambda}}$ contains an SPT rooted at s which is not a subgraph of H' iff there exists a shortest s-x path in $G_{\bar{\lambda}}$ that contains exactly one edge in $E \setminus E(H')$, for some $x \in V$. Therefore, we can state the following

Proposition 2. *Let H' be a subgraph of G containing some SPT of G rooted at s. The value $\delta(H')$ is equal to the minimum of the optimality thresholds of all the edges $e \in E \setminus E(H')$ in $(H' + e, s)$ w.r.t. SPT problem.* □

As a consequence, in order to compute $\gamma(e)$, it is sufficient to compute the optimality threshold of e in $H^* + e$ for each edge $e \in E \setminus E(H^*)$. For the rest of the proof assume that $e = (x, y) \in E \setminus E(H^*)$ is fixed. First of all, observe that all the s-x paths in $H^* + e$ are s-x shortest paths in G, so they are also shortest paths in $H^* + e$. Therefore, the structure H_1 of all the shortest s-x paths in $H^* + e$ computed by Algorithm 1 is equivalent to the structure of all the s-x paths of G. Moreover, Proposition 1 implies that Algorithm 1 makes only one iteration. As a consequence, the value `ub` returned by Algorithm 1 is the value B_1 computed during the first iteration.

To prove that $B_1 = \tau(e)$, let A be the set of vertices $v \in A_x \cap A_y$ for which a v-x path in H made up of vertices in $A_x \setminus A_y$ exists. Clearly, A is an s-x cut in H. Therefore, for every $v \in A$, $B_1 \geq \frac{d_{H_1}(v,x)+w(e)}{d_{H^*+e}(v,y)} = \texttt{cost}_1(v)$. Let $v^* \in A$ be a vertex such that for every $v \in A$, $\texttt{cost}_1(v^*) \leq \texttt{cost}_1(v)$. As for every $v \in A_x \cap A_y$ it is $\texttt{cost}_1(v) = \frac{d_{H_1}(v,x)+w(e)}{d_{H^*+e}(v,y)} = \frac{d_G(s,x)-d_G(s,v)+w(e)}{d_G(s,y)-d_G(s,v)}$, then v^* is a vertex of A that minimizes its distance from s in G. Let P be an s-x path in H^* passing through vertex v^*. We claim that the bottleneck value of P w.r.t. $\texttt{cost}_1$ is given by $\texttt{cost}_1(v^*)$ thus proving that $B_1 = \texttt{cost}_1(v^*)$. In fact, for every $v \in V(P[v^*, x])$, $v \neq v^*$, $\texttt{cost}_1(v) = 0$ as $d_{H^*+e}(v, y) = \infty$. Furthermore, for every $v \in V(P[s, v^*])$

$$\begin{aligned}\texttt{cost}_1(v) = \frac{d_{H_1}(v,x) + w(e)}{d_{H^*+e}(v,y)} &= \frac{d_G(v,v^*) + d_G(v^*,x) + w(e)}{d_G(v,v^*) + d_G(v^*,y)} \\ &\leq \frac{d_G(v^*,x) + w(e)}{d_G(v^*,y)} \\ &= \texttt{cost}_1(v^*),\end{aligned}$$

as $\texttt{cost}_1(v^*) \geq 1$ and $d_G(v, v^*) \geq 0$.

Therefore, to prove $B_1 = \tau(e)$, it is enough to prove that $\tau(e) = \texttt{cost}_1(v^*)$, or equivalently, that $\min_{v \in A_x \setminus A_y} \eta_v = d_G(s, v^*)$. But this is immediately true by definition of η_v since A is an s-v cut in H^* for every $v \in A_x \setminus A_y$ and edge weights are non negative. This completes the proof. □

5 The Shortest Path Stability Problem

Let $G = (V, E, w)$ be an edge-weighted directed graph with $|V| = n$, and $|E| = m$ (we assume that $w(e) > 0$ for every edge $e \in E$), and let $s, t \in V$ be two vertices of G. The *shortest path (SP)* problem asks for computing a path form s to t of minimum length. In this section we provide an efficient algorithm to solve Q_1^* when all the shortest $s-t$ paths are vertex disjoint (except for the vertices s and t). Let H be the subgraph of G induced by all these shortest paths, and let σ denote the stability number of G (w.r.t. the SP problem with source s and destination t), i.e. $\sigma = \delta(H)$.

From now on we will assume that $P_1, \ldots, P_k$ are all the shortest $s-t$ paths in G. For the sake of simplicity, we will assume that G always contains an $s-t$ path P which is not a shortest path, otherwise $\sigma = +\infty$.

The algorithm considers a set of non-optimal paths having a specific form (as we will specify soon) and for each of them, say P, it computes a value $\theta(P)$, which is defined as follows: $\theta(P)$ is the minimum value such that there exists a $\theta(P)$-perturbation $\bar{\lambda}$ in which P is not longer than every P_i, $i = 1, \ldots, k$, i.e. $w(P, \bar{\lambda}) \leq w(P_i, \bar{\lambda})$, $i = 1, \ldots, k$. Notice that by definition of σ, the following property holds:

Remark 1. $\sigma \leq \theta(P)$, for any non-optimal P.

Our algorithm uses a key property which is stated in the lemma below. Roughly speaking, the main idea is that there must exist an $s-t$ path P with $\theta(P) = \sigma$ and such that P has a very nice form, namely P consists of three pieces, two of which (the first and the third one) are subpaths of some shortest $s-t$ paths in G, while the central piece is a shortest path which is vertex disjoint from H (except for the first and the last node). An example is shown in Figure 3, where the path P is the one passing through nodes s, u, v', v, t and where its central part consists of the subpath going from u to v. Moreover, notice that in this example $P[u, v]$ is not a shortest path in G. This shows that the vertex disjointness property of the central piece is actually needed. More formally:

Lemma 5. *There exists a non-optimal path P having $\theta(P) \leq \sigma$ and such that:*

1. *$P = P_i[s, u] \cup P[u, v] \cup P_j[v, t]$, for some $u, v \in V, i, j = 1, \ldots, k$;*
2. *$P[u, v]$ is a shortest path from u to v which is vertex disjoint from H (except for u and v).*

Proof. Let us consider an edge $\tilde{e} \in E \setminus E(H)$ with $\gamma(\tilde{e}) = \sigma$. Then, there must exist a path P' which is a shortest path from s to t in some σ-perturbation of G. Clearly, P' is not a shortest path in G. W.l.o.g. we can decompose P' as follows: $P' = P_i[s, u] \cup P'[u, v] \cup P'[v, t]$ where $P'[u, v]$ is vertex disjoint from H (except for u and v). Let $\bar{\lambda}$ be the σ-perturbation defined as $\bar{\lambda}_e = 1$ if $e \in E(P')$, σ otherwise. Notice that $w(P', \bar{\lambda}) \leq w(P_\ell, \bar{\lambda})$ for every $\ell = 1, \ldots, k$.

Then, we define a new path $P = P_i[s, u] \cup P[u, v] \cup P_j[v, t]$, where $P[u, v]$ is a shortest path from u to v that is vertex disjoint from H and P_j is the path v

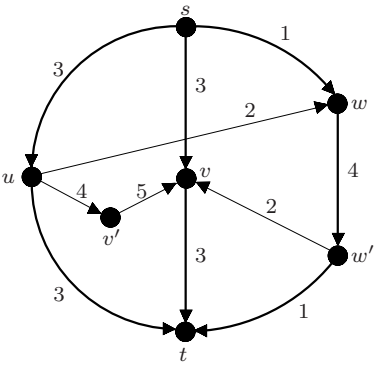


Fig. 3. An instance of Q_1^*. The structure H of all shortest $s-t$ paths is in bold. The path P with $\theta(P) = \sigma$ is $s \to u \to v' \to v \to t$. The stability number of H is 4. To see this, notice that if we perturb all edges but P by a value strictly greater than 4, P becomes the only $s-t$ shortest path (hence, $\sigma \leq 4$). On the other hand, any other $s-t$ path using the edge (u,w) or the edge (w',v) cannot become minimum without perturbing by a factor 5 the edge (s,w) or the edge (w',t), respectively.

belongs to. Now, let $\bar{\lambda}'$ be the σ-perturbation defined as $\bar{\lambda}$ except that $\bar{\lambda}'_e = 1$ for every $e \in E(P[u,t])$. We show that $\theta(P) \leq \sigma$ by proving that the length of P in $\bar{\lambda}'$ is at most that of P_ℓ, for each $\ell = 1, \ldots, k$.

We have that:

$$\begin{aligned} w(P,\bar{\lambda}') &= w(P_i[s,u],\bar{\lambda}') + w(P[u,v],\bar{\lambda}') + w(P_j[v,t],\bar{\lambda}') \\ &= w(P_i[s,u],\bar{\lambda}) + w(P[u,v],\bar{\lambda}') + w(P_j[v,t],\bar{\lambda}') \\ &\leq w(P_i[s,u],\bar{\lambda}) + w(P'[u,v],\bar{\lambda}) + w(P_j[v,t],\bar{\lambda}') \\ &= w(P'[s,u],\bar{\lambda}) + w(P'[u,v],\bar{\lambda}) + w(P_j[v,t],\bar{\lambda}') \\ &= w(P'[s,v],\bar{\lambda}) + w(P_j[v,t],\bar{\lambda}'). \end{aligned} \tag{1}$$

As a consequence, since $P_j[v,t]$ is a shortest path in the original graph, we have $w(P_j[v,t],\bar{\lambda}') \leq w(P'[v,t],\bar{\lambda})$, and thus $w(P,\bar{\lambda}') \leq w(P',\bar{\lambda}) \leq w(P_\ell,\bar{\lambda})$ for every $\ell = 1,\ldots,k$. Moreover, for every $\ell \neq j$, it holds that $w(P_\ell,\bar{\lambda}) = w(P_\ell,\bar{\lambda}')$. Hence, we have $w(P,\bar{\lambda}') \leq w(P_\ell,\bar{\lambda}')$ for every $\ell \neq j$. On the other hand, as far as P_j is concerned, from (1), since $w(P'[s,v],\bar{\lambda}) \leq w(P_j[s,v],\bar{\lambda}) = w(P_j[s,v],\bar{\lambda}')$, we have $w(P,\bar{\lambda}') \leq w(P_j,\bar{\lambda}')$. □

The algorithm to compute σ works as follows: it checks all paths of the form of Lemma 5 and selects a path with minimum θ value among them. More precisely, for each ordered pair of nodes $u \in P_i$ and $v \in P_j$ (where i and j may coincide), the algorithm considers the path $P_{u,v} = P_i[s,u] \cup P[u,v] \cup P_j[v,t]$, where $P[u,v]$ is a shortest path from u to v which is vertex disjoint from H (except for u and v), then it computes $\theta(P_{u,v})$, and finally it takes the minimum $\theta(P_{u,v})$ value over all pairs u, v.

The correctness of the algorithm immediately follows from Lemma 5 and Remark 1. Moreover, it is not too hard to see that:

$$\theta(P_{u,v}) = \begin{cases} \max\left\{\frac{w(P_{u,v}[u,t])}{w(P_i[u,t])}, \frac{w(P_{u,v}[s,v])}{w(P_j[s,v])}\right\} & \text{if } i \neq j; \\ \frac{w(P_{u,v}[u,v])}{w(P_i[u,v])} & \text{otherwise.} \end{cases}$$

We are now ready to prove the following:

Theorem 4. *The shortest $s-t$ path stability problem can be solved in $\mathcal{O}(mn + n^2 \log n)$ time if the shortest paths from s to t are pairwise vertex disjoint.*

Proof. First, we compute H in $\mathcal{O}(m + n \log n)$ time. Then, we compute all the lengths of the $P_{u,v}[u, v]$ paths in $\mathcal{O}(mn + n^2 \log n)$ as follows. For each node v, we compute in $\mathcal{O}(m + n \log n)$ time a shortest vertex disjoint path from v to every u by modifying G as follows: for each node u, we delete all the edges in $E(H)$ outgoing from u, and then we run Dijkstra's algorithm to find a shortest path tree rooted at v in the obtained graph. Finally, we can compute $\theta(P_{u,v})$ in constant time. Since the number of paths to be considered are at most n^2, the claim follows. □

References

1. Beyer, H.-G., Sendhoff, B.: Robust optimization - a comprehensive survey. Computer Methods in Applied Mechanics and Engineering 196(33-34), 3190–3218 (2007)
2. Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Comput. 21(6), 1184–1192 (1992)
3. Gal, T., Greenberg, H.J. (eds.): Advances in sensitivity analysis and parametric programming. Int. Series in Operations Research and Management Science, vol. 6. Kluwer Academic Publishers, Boston (1997)
4. Kouvelis, P., Yu, G.: Robust discrete optimization and its applications. Kluwer Academic Publishers, Dordrecht (1997)
5. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM 49(1), 16–34 (2002)
6. Pettie, S.: Sensitivity analysis of minimum spanning trees in sub-inverse-Ackermann time. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 964–973. Springer, Heidelberg (2005)
7. Shier, D.R., Witzgall, C.: Edge tolerances in shortest path and network flow problems. Networks 10(4), 277–291 (1980)
8. Tarjan, R.E.: Sensitivity analysis of minimum spanning trees and shortest path trees. Inf. Proc. Letters 14(1), 30–33 (1982)

Space Complexity of Self-stabilizing Leader Election in Passively-Mobile Anonymous Agents*

Shukai Cai, Taisuke Izumi, and Koichi Wada

Graduate School of Engineering, Nagoya Institute of Technology, Nagoya, 466-8555, Japan
csk@phaser.elcom.nitech.ac.jp, {t-izumi,wada}@nitech.ac.jp

Abstract. A population protocol is one of distributed computing models for passively-mobile systems, where a number of agents change their states by pairwise interactions between two agents. In this paper, we investigate the solvability of the self-stabilizing leader election in population protocols without any kind of oracles. We identify the necessary and sufficient condition to solve the self-stabilizing leader election in population protocols from the aspects of local memory complexity and fairness assumptions. This paper shows that under the assumption of global fairness, no protocol using only $n - 1$ states can solve the self-stabilizing leader election in complete interaction graphs, where n is the number of agents in the system. To prove this impossibility, we introduce a novel proof technique, called closed-set argument. In addition, we propose a self-stabilizing leader election protocol using n states that works even under the unfairness assumption. This protocol requires the exact knowledge about the number of agents in the system. We also show that such knowledge is necessary to construct any self-stabilizing leader election protocol.

1 Introduction

A *passively-mobile* system is a collection of agents that move in a certain region but have no control over how they move. Since the communication range of each agent is quite small compared to the size of the region, two agents can communicate only when they are sufficiently close to each other. Passive mobility appears in many real systems. A representative example is a network of smart sensors attached to cars or animals. In addition, a certain kind of natural computing, such as synthesis of chemical materials and complex biosystems, can be included in passively-mobile systems by regarding chemical interactions as communications. While these systems are different in the view of applications, all of them aim to a common goal, that is, how to organize and manipulate computing entities that are uncontrollable in the sense of mobility. Then, it is reasonable to think about some common principles underlying them. Revealing such principles from the aspect of theoretical computer science is an interesting and worthwhile challenge.

* This work is supported in part by the Japan Society for the Promotion of Science: Grant-in-Aid for Young Scientists(B)(19700058), the Japan Society for the Promotion of Science: Grant-in-Aid for Scientific Research(C)(21500013) and Hori Information Science Promotion Foundation.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 113–125, 2010.

Recently, as a model for such passively-mobile systems, *population protocols* are introduced [1,2,7]. A population protocol consists of a number of agents, to which some program (protocol) is deployed. Following the deployed protocol, each agent changes its state by *pairwise interactions* to other agents (that is, two agents come closer to each other in the region and update their states by exchanging information). Typically, the capability of each agent is limited. It is often assumed that each agent has only constant-space memory and no identifier. A population protocol is a good abstraction that captures the feature of passively-mobile systems in spite of its mathematical simplicity. [3,4,5,6,8,10].

Population protocols are originated by Angluin et al. [1], which investigates a class of predicates that can be computed autonomously over population protocols. Its primary result is that any predicate in *semilinear* class (which includes the comparison, modulo and threshold predicates) can be computed on population protocols by proposing a protocol that stably computes any semilinear predicate. In the following paper [4], it is also shown that any computable predicate by population protocols belongs to semilinear, that is, semilinear is the necessary and sufficient class of the predicates that can be computed on population protocols of all-pairs interaction graphs (complete interaction graphs).

The protocols proposed in the above paper are assumed to start from a properly-formed system configuration. In this sense, it is not a *self-stabilizing* protocol: Self-stabilization is one of the desirable properties of distributed computations, which ensures that the system necessarily converges to the desired behavior regardless of its initial configuration. Self-stabilization on population protocols is considered in a number of previous papers [2,9,11], which have investigated the solvability of the *self-stabilizing leader election* (SS-LE) problems under some kinds of assumptions. The general model of population protocols introduces an *interaction graph*, which specifies the possibility of communication between two agents. The above papers show the solvability and unsolvability of SS-LE for specific classes of interaction graphs such as complete graphs, rings, rooted trees, directed acyclic graphs, and so on. Unfortunately, it is easily shown that SS-LE is almost impossible in general. Thus, the above papers also consider some additional (but reasonable) assumptions to make SS-LE solvable by introducing several notions extending the computational power of population protocols: *global fairness* and *leader detector* oracle $\Omega?$. Intuitively, global fairness guarantees the occurrence of any possible transition and thus it prevents livelock caused by some looped execution. The leader detector oracle is an abstracted virtual device that informs the existence and inexistence of a leader to all the agents in the system. Both of the assumptions give some additional computational power to population protocols, which is sufficient to solve SS-LE in some cases, but insufficient in some other cases. However, the complete characterization of system assumptions making SS-LE solvable is unknown. Currently, only a few results about the solvability of SS-LE on the complete interaction graphs are known:

1. Assuming global fairness and the oracle $\Omega?$, there exists an SS-LE protocol where each agent uses only one bit of memory [11].
2. Under the assumption of unfairness and no oracle, no uniform protocol can solve SS-LE, where "uniform protocol" means the one that works correctly on the system

with arbitrary number of agents (that is, uniform protocols do not use any information about the total number of agents) [2].
3. Without $\Omega?$, any protocol using only one bit of memory cannot solve SS-LE even if we assume global fairness [9].

In this paper, we also investigate the solvability of SS-LE on population protocols. In particular, we are interested in self-stabilizing leader election protocols in complete interaction graphs without oracles. The primary contribution of our work is to identify the necessary and sufficient conditions such that SS-LE becomes solvable from the aspects of local memory space and fairness assumptions. More precisely, this paper shows the following three results:

1. Without oracles, there is no deterministic or probabilistic SS-LE protocol using only $n - 1$ states of memory even if we assume global fairness, where n is the number of agents in the system.
2. There exists an SS-LE protocol that uses n states ($\lceil \log_2 n \rceil$ bits) of memory and correctly works under the unfairness assumption.
3. Even if we assume global fairness, without oracles, there is no uniform SS-LE protocol in the strong sense. That is, any SS-LE protocol working correctly on the population of n agents does not work correctly on the population of $n - 1$ agents.

The third result implies that the upper bound for the number of agents is not sufficient knowledge to design SS-LE protocols, and thus it justifies the fact that the exact value of n is necessary to construct the protocol shown in the second possibility result. It should be also noted that the first impossibility result is quite nontrivial and interesting. Global fairness is reasonable but sufficiently strong so that it can break essential ideas leading previous impossibility results. Actually, under the global fairness assumption, we cannot apply many of existing techniques to prove the impossibility. In this paper, we resolve such difficulty by introducing a novel proof technique based on *closed sets*. Our key idea is to identify the set of states that never creates the leader state. While this paper utilizes this technique to show the impossibility of SS-LE, we believe that it can be applied to more broader cases, including other problems and other graph classes, to prove the impossibility under the global fairness assumption. Moreover, we can show that the three results are all correct for both the traditional two-way protocol and the one-way protocol (the two-way protocol allows that two agents can change both of their states in the interaction, but the one-way one does not). That is, the impossibility result holds in the stronger two-way protocol and the possibility result even holds in the weaker one-way protocol.

1.1 Related Work

Leader election on population protocols are first introduced in [3]. In [2], a non-uniform population protocol is given to solve the self-stabilizing leader election problem in directed rings of odd size under the assumption of global fairness. The authors also show that there is no uniform self-stabilizing leader election protocol for any non-simple class of interaction graphs, where a class C is non-simple if any graph in C can be partitioned into two subgraphs belonging to C.

In [11], Fischer and Jiang introduce *eventual leader detector* $\Omega?$ to realize uniform self-stabilizing leader election protocols. They give a uniform SS-LE protocol for complete graphs under the weaker fairness assumption than global one (it is called *local fairness* and the most usual assumption of fairness) using only 1 bit of memory, and a uniform self-stabilizing leader election protocol for directed rings under the assumption of global fairness. It is also shown that there exists no uniform self-stabilizing leader election protocol for directed rings under the local fairness assumption. All the above results are obtained with the help of $\Omega?$.

Canepa and Gradinariu [9] investigates the feasibility of one-bit protocols: They give a uniform one-bit SS-LE protocol for rooted trees and acyclic graphs with only one sink-node. Also they give a probabilistic protocol for arbitrary graphs under local fairness. Moreover, they prove $\Omega?$ is necessary to realize uniform one-bit SS-LE protocols for any class of interaction graphs. All the results in the paper are under the assumption of using 1 bit of memory and with the help of $\Omega?$.

2 Model and Definitions

We introduce the formal definitions of population-protocol considered in this paper.

A population consists of n agents, which can change their own states by interacting with each other. In the general model of population protocols, all pairs of agents do not necessarily have direct interactions. The possibility of direct interactions between two agents is specified by interaction graphs: An interaction graph G=(V, E) is a simple directed graph where each vertex, labeled by $v_1, v_2, v_3, \cdots$, corresponds to each agent. The edge from v_i to v_j implies that the agent corresponding to v_i can interact to the agent for v_j, where v_i is the *initiator* and v_j is the *responder*. Throughout this paper, we assume that the interaction graph is complete. That is, any pair of agents is possible to interact with each other. For convenience, we use undirected complete graphs for the bidirectional completed graphs in what follows.

A *protocol* $P = (Q, \delta)$ is a pair of a finite set Q of *states* and a *transition function* δ that maps each pair of states $Q \times Q$ to a nonempty subset of $Q \times Q$. The transition function, and the protocol, is *deterministic* if $\delta(p, q)$ always contains just one pair of states. Otherwise the protocol is called a *nondeterministic* protocol. For convenience, in this paper, we only consider deterministic protocols, and thus we simplify the definition of a transition function to a mapping $\delta : Q \times Q \to Q \times Q$ (i.e., the states after each transition is uniquely determined). If the two agents involved in an interaction can learn the states of each other and change their states depending on the state of the other, we call the protocol a *two-way* one. By contrast, if the initiator has no chance to change its state and only the responder can change its state after an interaction, we call the protocol a *one-way* one. In the one-way protocol, for any transition $r : (p, q) \to (p', q')$, $p' = p$ for any $p \in Q$. Notice that a transition does not necessarily cause either of the nodes to change its state. That is, a transition $(p, q) \to (p, q)$ is possible. We define *silent* transitions as ones that do not change any state. The transition that is not silent is said to be *active*.

From the definition of the two-way and the one-way protocols, it is obviously that the one-way protocol is a special case of the two-way one. Thus the computational power

of the one-way protocol is not stronger than the computational power of the two-way one. More precisely, the one-way protocol is also correct for the two-way one, and an unsolvable problem for the two-way protocol is still unsolvable for the one-way protocol.

Formally, a configuration C is an n-tuple $(q_1, q_2, q_3, \cdots, q_n)$ of states where each entry q_k corresponds to the state of the agent v_k. The state of an agent v_k at the configuration C is denoted by $C(v_k)$. Letting C be a configuration, and r be a transition that maps (p, q) to (p', q'), we say that r is *enabled* in C if there exists an edge (v_i, v_j) such that $C(v_i) = p$ and $C(v_j) = q$. Then, we say that C can go to C' via r, denoted by $C \xrightarrow{r} C'$, if C' is the configuration that is obtained by changing the states of v_i and v_j to p' and q', respectively. We simply say that C can go to C', denoted $C \to C'$, if $C \xrightarrow{r} C'$ holds for some transition r. We define executions in population protocols as follows:

Definition 1 (Execution). *Letting $P = (Q, \delta)$ be a protocol, an* execution *of P is an infinite sequence of configurations and transitions $C_0, r_0, C_1, r_1, \cdots$ satisfying*

1. *for each i, r_i is a transition of δ and $C_i \xrightarrow{r_i} C_{i+1}$, $i = 0, 1, \cdots$ holds , and*
2. *r_i is active for infinitely many i unless all the enabled transitions are silent.*

Notice that the second condition ensures the progress of protocols (i.e., it excludes the meaningless executions such that only silent transitions appear).

2.1 Fairness Assumption

Fairness is an assumption that restricts the behavior of systems. Formally, it is defined as a constraint for executions. In this paper, we introduce the following fairness assumptions [11]:

Definition 2 (Global fairness assumption $\mathbb{G}$). *An execution $E = C_0, r_0, C_1, r_1, \cdots$ is globally fair: for every C and C' such that $C \to C'$, if $C = C_i$ for infinitely many i, then $C_i = C$ and $C_{i+1} = C'$ for infinitely many i.*

Intuitively, global fairness guarantees the possibility of the occurrence of any possible execution and thus it prevents the occurrence of livelock caused by some looped execution.

By contrast to global fairness, the assumption called local fairness is usually used. The local fairness only guarantees that each transition can be taken infinitely often is actually taken infinitely often.

In addition to the above, we also define *unfairness assumption* $\mathbb{U}$, which requires no assumption to executions. Given a protocol P and a fairness assumption $X \in \{\mathbb{G}, \mathbb{U}\}$, we define $\mathcal{E}_X(P)$ be the set of all executions of P satisfying the fairness assumption X.

2.2 Self-stabilization, Legitimate Configurations

Self-Stabilizing protocols guarantee the convergence to their desired behavior starting from any initial configuration. In this paper, we consider the self-stabilizing leader election over populations, which requires that the system eventually reach a *legitimate configuration*, where exactly one process keeps a special state, called *leader state*, and no other leader state is generated in any following execution. Formally, the self-stabilizing leader election problem is defined as follows:

Definition 3 (Self-stabilizing leader election). *A protocol P solves the self-stabilizing leader election under the fairness assumption X if there is one special state s and any execution E in $\mathcal{E}_X(P)$ satisfies that there exist some i and v_k such that for any $j \geq i$ and $h \neq k$, $C_j(v_k) = s$ and $C_j(v_h) \neq s$ hold.*

3 Impossibility of Self-stabilizing Leader Election Using $n - 1$ States

In this section, we will show that without the help of $\Omega?$, any self-stabilizing leader election two-way protocol is impossible in a complete network graph under global fairness using only distinct $n - 1$ states.

3.1 Difficulty of Proving Impossibility under Global Fairness

In this subsection, we explain why it is a quite nontrivial and difficult task to prove impossibility under global fairness. We show that existing techniques used to prove the impossibility do not work under the global fairness assumption.

Roughly speaking, most of existing impossibility proofs for SS-LE are roughly divided into two types: One is the argument by *illegal loop*, and the other one is that by *partition*. We explain the details for both of them:

Illegal loop argument: The key idea of the illegal loop argument is to find a looped execution including a non-legitimate configuration. The infinite execution repeating the loop never converges to legitimate configurations, which contradicts the self-stabilization property. This kind of arguments is widely used in almost all areas of distributed computation. However, it cannot be applied to prove the impossibility under global fairness because the global fairness assumption does not allow the system to periodically repeat the same behavior: If the system does such looped behavior, any configuration in the loop appears infinitely often. Then, under global fairness, it is necessarily guaranteed that the system could escape from the looped execution if there exists a transition which can lead the system to exit from the looped execution.

Partition argument: Partition argument is the technique using the fact that it is difficult to break a certain kind of symmetry. The basic idea of the partition argument is to divide a given n-node interaction graph into two same subgraphs with size $n/2$ (in general, division to three or more subgraphs can be considered). By their symmetry, it is possible to show the existence of the execution that converges to the configuration where the two subgraphs independently and separately elect a leader respectively. Thus, it contradicts the uniqueness of leaders. However, this argument can be applied only to the case of uniform protocols because non-uniform protocols do not guarantee to elect one leader in the divided subgraph (that is, it is not guaranteed that the protocol works correctly on $n/2$ agents). Moreover, to make an execution where two subgraphs independently elect a leader respectively, we have to prohibit the interactions between the two subgraphs. However, if some interaction is enabled on an edge that joints two subgraphs infinitely often, it must occur necessarily under global fairness. We cannot eliminate the

possibility that such interaction breaks the symmetry, and the system converges to the legitimate configuration.

To circumvent the problems which the above two arguments hold, in the following subsection, we newly introduce a proof technique based on closed sets. Intuitively, the closed set argument finds a set of states such that the interactions between any pair of two states in the set create no state out of the set. The key of our proof is to find a closed set excluding the leader state and obtain a contradiction.

3.2 Impossibility Using $n - 1$ States

First, we introduce several notions necessary for the following proofs.

For convenience of the proof for the impossibility, we extend the definition of a configuration. A configuration C is an n-tuple of states of agents or $\perp$, where $\perp$ is a special value that masks the state of the corresponding agent. For example, $C = (\perp, q_2, q_3, \cdots, q_n)$ is also a configuration. The size of a configuration C is the number of non-$\perp$ values appearing in C, and it is denoted by $|C|$. A *subconfiguration* C' of a configuration C is an n-tuple obtained by replacing several entries in C by $\perp$. For example, letting $C = (a, b, d, e)$ be a configuration, $C'_1 = (a, \perp, d, e)$, $C'_2 = (\perp, \perp, d, \perp)$, and $C'_3 = (\perp, b, d, \perp)$ are subconfigurations of C whose sizes are 3, 1, and 2, respectively. In addition, C'_2 is also a subconfiguration of C'_1 and C'_2 itself, but C'_3 is not a subconfiguration of C'_1.

A *trace* is a sequence of transitions. We say a trace $T = r_1, r_2, \cdots, r_i$ is *applicable* to a configuration C_0 if there exists a sequence of configurations $C_0, C_1, \cdots, C_i$ such that $C_0 \xrightarrow{r_1} C_1 \xrightarrow{r_2} C_2 \xrightarrow{r_3} \cdots \xrightarrow{r_i} C_i$. We define the length of a trace T as the number i of transitions appearing in T. For a configuration C and a trace T applicable to C, we define $\sigma_T(C)$ as the configuration resulted by applying T to C. If $C' = \sigma_T(C)$ holds, we often use the notation $C \xrightarrow{T} C'$.

A configuration C' is *reachable* from a configuration C, denoted by $C \xrightarrow{*} C'$, if there exists a trace T such that $C \xrightarrow{T} C'$. We say a configuration C can *generate* state p, if there is a configuration C' that is reachable from C and contains p. See Figure 1. For a set G of states, if a configuration C cannot generate any state in G, we say C cannot generate G. Letting $P = (Q, \delta)$ be a population protocol, a subset G of Q is called a *closed* set of P if for any transition $r : (p, q) \to (p', q')$ in δ, $p, q \in G$ implies $p', q' \in G$.

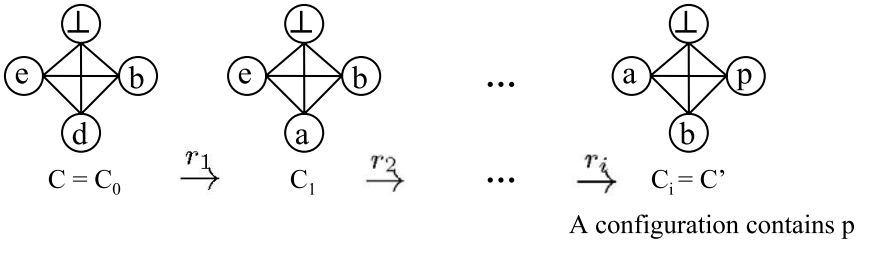


Fig. 1. A configuration C can generate state p

We first show three fundamental lemmas obtained from the above definitions.

Lemma 1. *Let C' be a subconfiguration of C. If a trace T is applicable to C', it is also applicable to C, and $\sigma_T(C')$ is a subconfiguration of $\sigma_T(C)$.*

Proof. Let t be the length of T (i.e., the number of transitions appearing in T). The lemma is proved by the induction on the length of T.

(Basis) $t = 1$: Let $r : (p, q) \to (p', q')$ be a transition appearing in T. Then, clearly, if r is enabled in C', it is also enabled in C. This implies that T is applicable to C. Let u and v be the initiator and responder of the transition r, since both $\sigma_T(C')$ and $\sigma_T(C)$ are the configurations obtained from C' and C by replacing the states of u and v by p' and q' respectively. Thus, $\sigma_T(C')$ is a subconfiguration of $\sigma_T(C)$.

(Inductive Step): Suppose that the lemma holds for any trace with length $t - 1$ or less. Then, we split the trace T into two traces T_1 and T_2 (T_1 is a prefix of T and T_2 is the remaining part). Then, since the length of T_1 is less than t, by the induction hypothesis, we can conclude T_1 is applicable to C and $\sigma_{T_1}(C')$ is a subconfiguration of $\sigma_{T_1}(C)$. It is clear that T_2 is applicable to $\sigma_{T_1}(C')$ because T is applicable to C'. Thus, again by the induction hypothesis, T_2 (whose length is less than t) is applicable to $\sigma_{T_1}(C)$ and $\sigma_{T_2}(\sigma_{T_1}(C'))$ is a subconfiguration of $\sigma_{T_2}(\sigma_{T_1}(C))$. This implies the lemma holds. □

Lemma 2. *If a configuration C cannot generate a set of states G, then for any configuration C' such that $C \xrightarrow{*} C'$, C' cannot generate G.*

Proof. Suppose for contradiction that C' can generate a state p in G. Then, there exists a configuration D such that $C' \xrightarrow{*} D$ and $p \in D$. Since $C \xrightarrow{*} C'$ holds, we obtain $C \xrightarrow{*} D$, which contradicts the fact that C cannot generate G. □

Lemma 3. *If a configuration C cannot generate a set of states G, any subconfiguration of C cannot generate G.*

Proof. Suppose for contradiction that a subconfiguration C' of C can generate a state p in G. Then, there exists a trace T such that $\sigma_T(C')$ includes the state p. By Lemma 1, T is also applicable to C and $\sigma_T(C')$ is a subconfiguration of $\sigma_T(C)$. This implies p belongs to $\sigma_T(C)$, which is contradiction. □

The following lemmas are the keys of our impossibility result.

Lemma 4. *Let G ($|G| < n - 1$) be a set of states, and C ($|C| > 0$) be a configuration that cannot generate G. Then, either of the following conditions holds:*

1: *The complement of G (denoted by $\bar{G}$) is closed.*
2: *There exist a configuration C' and a superset G' of G such that $|C| - 1 \leq |C'|$ and $|G| + 1 = |G'|$ hold, and C' cannot generate G'.*

Proof. We prove this lemma by showing that the condition 2 necessarily holds if the complement of G is not closed. Assuming that $\bar{G}$ is not closed, there exists a transition $r : (p, q) \to (p', q')$ such that $p, q \notin G$ and at least one of p' and $q' \in G$ (because if such a transition does not exist, any interaction of two states in $\bar{G}$ results in two states in $\bar{G}$, which implies that $\bar{G}$ is closed). Then we consider the following two cases:

1. One of p and q cannot be generated by C: Without loss of generality, we assume that C cannot generate p. Then, C cannot generate $\{p\} \cup G$. Therefore, we obtain $C' = C$ and $G' = G \cup \{p\}$ satisfying the condition 2.
2. Both of p and q can be generated by C: Since C can generate p, there exists a configuration D such that $C \xrightarrow{*} D$ and $p \in D$. We consider the subconfiguration D' that is obtained by replacing the entry of p in D by $\perp$. Then, if we can show that D' cannot generate q, the lemma is proved by letting $C' = D'$ and $G = G \cup \{q\}$. In the following, we show it actually holds: Suppose for contradiction that D' can generate q. Then, there exists a trace T that makes D' reach a configuration with q. By Lemma 1, T is also applicable to D, and $\sigma_T(D)$ includes both p and q. This implies that C can reach the configuration $\sigma_T(D)$ that includes both p and q. Then, It is clear that C can generate both p' and q' because the transition r is enabled in the configuration $\sigma_T(D)$. However, either of p' or q' belongs to G and thus it is contradict to that C cannot generate G. □

Lemma 5. *Any self-stabilizing leader election protocol P has no closed set excluding its leader state.*

Proof. Suppose for contradiction that P has a closed set H which excludes its leader state in P. Consider an initial configuration C whose states are all in H. Since H is closed, so C can only generate the states in H. Because the leader state is not in H, C cannot generate a leader state. This implies that any execution starting from C cannot reach a configuration with leader. It is contradiction. □

By using the above two lemmas, we can show the impossibility of self-stabilizing leader election using only $n - 1$ states.

Theorem 1. *There is no self-stabilizing leader election protocol that uses only $n - 1$ states.*

Proof. We assume for contradiction that a self-stabilizing leader election protocol P which uses only distinct $n-1$ states. The $n-1$ states of the protocol P are denoted by $Q = \{s_0, s_1, s_2, \cdots, s_{n-2}\}$, where s_0 is the leader state. The set of all transitions constituting $\mathcal{P}$ is denoted by $\delta_{\mathcal{P}}$.

Letting C be a legitimate configuration, that is, exactly one leader exists in it and another leader is not newly created in any following execution. This implies that the subconfiguration C' which obtained by masking the leader state s_0 in C cannot generate the leader state s_0. Thus, letting $C_0 = C'$ and $G_0 = \{s_0\}$, C_0 cannot generate G_0, and it holds that $|C_0| = n - 1$ and $|G_0| = 1$. By Lemma 5, there is no closed set excluding s_0 in P. Thus, the complement of G_0 is not closed. Then, by Lemma 4, we can obtain a configuration C_1 and a superset G_1 of G_0 satisfying that $|C_1| \geq |C_0| - 1 = n - 2$, $|G_1| = |G_0| + 1$, and C_1 cannot generate G_1. Similarly, we can also obtain C_{i+1} and G_{i+1} from C_i and G_i by applying Lemma 4 repeatedly. Finally, after applying the lemma $n-2$ times, we have a configuration C_{n-2} and a set G_{n-2} satisfying $|C_{n-2}| \geq 1$, $|G_{n-2}| = n - 1$ and C_{n-2} cannot generate G_{n-2}. See Figure 2. Then, G_{n-2} is equivalent to Q, and thus C_{n-2} cannot generate any state. However, C_{n-2} is not empty, which implies a state s_x $(0 \leq x \leq n - 2)$ in C_{n-2} can be generated by C_{n-2}. This is contradiction. □

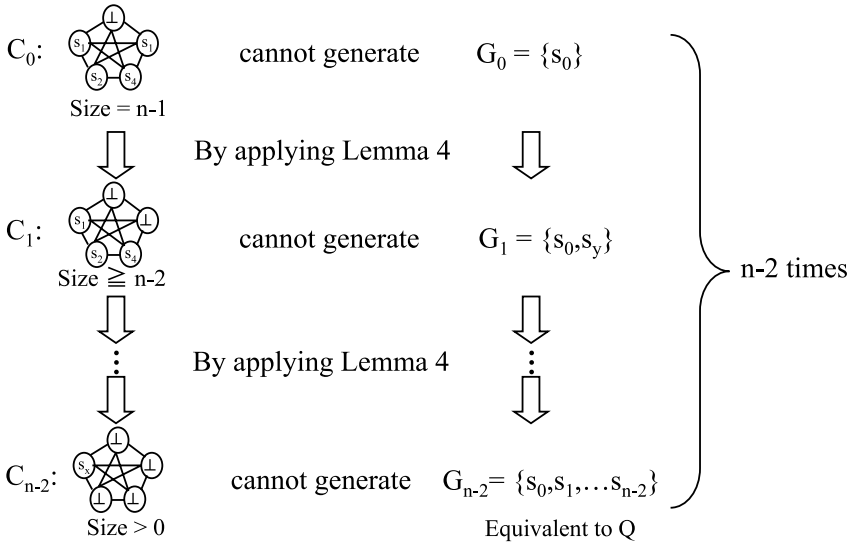


Fig. 2. Prove the theorem by contradiction

Since the one-way protocol is a special case of the two-way one, the impossibility result holds even for the one-way protocol. And any impossibility result for the two-way protocol also holds in the one-way one.

Remarks 1. *Noting that our proof does not request any constraint to the transition function. That means the impossibility result also holds for a probabilistic protocol. Thus, our impossibility result can be extended as follows: Without oracles, there is no deterministic or probabilistic SS-LE protocol using only* $n-1$ *states of memory even if we assume global fairness.*

4 Leader Election Protocol Using *n* States

In this section, we will show a self-stabilizing leader election one-way protocol which uses distinct n states. The n states of the protocol are denoted by $Q=\{s_0, s_1, s_2, \cdots, s_{n-1}\}$, where s_0 is the leader state. The proposed protocol is quite simple: When two agents with the same state interact, the responder will increment the subscript of its state (modulo n). That is, when the state of the responder is s_i, it will changed to be s_{i+1}, $i = 0, 1, \cdots, n-2$, exceptionally, s_{n-1} will be changed to s_0.

Protocol 1 $(s_i, s_i) \rightarrow (s_i, s_{(i+1) \mod n})$, $(i = 0, 1, \cdots, n-1)$

In what follows, we show that the above protocol correctly elects a unique leader. First, we introduce several notions necessary for the proofs. Throughout this section, we use another representation of each configuration $C = (m_0(C), m_1(C), \cdots, m_{n-1}(C))$, where $m_k(C)$ $(0 \leq k < n)$ is the number of agents having the state s_k in C. We also define $\#_0(C)$ to be the number of $m_k(C)$ $(k = 0, 1, \cdots, n-1)$ such that $m_k(C) = 0$ holds.

Lemma 6. *A configuration C with $\#_0(C) = 0$ is a legitimate configuration.*

Proof. From the definition of $\#_0(C)$, we know $\#_0(C) = 0$ means: for every s_k ($k = 0, 1, \cdots, n-1$), there exists an agent whose state is s_k in C. Because the number of agents equals to the number of states, so the states of every agents are different. Therefore, in any following execution, there is only one leader state s_0 in the configuration and the state of each agent will not be changed. □

The correctness of the protocol is proved by the argument based on monotonically-decreasing function, which is a standard technique for proving the correctness of self-stabilizing protocols. We first define the *distance* between two states.

Definition 4 (Distance Function). *For any configuration C, the distance $d_{k,j}(C)$ from the state s_k to s_j is defined as follows:*

$$d_{k,j}(C) = \begin{cases} 0 & (m_j(C) \neq 0 \text{ or } k = j) \\ (j-k)(m_k(C)-1) & (0 \leq k < j) \\ (j+n-k)(m_k(C)-1) & (j < k \leq n) \end{cases}$$

The total distance $d_j(C)$ *of state s_j in C is the sum of the distances from any state to s_j, that is, $d_j(C) = \Sigma_{k=0}^{n-1} d_{k,j}(C)$.*

From the definition, a pair of different states (s_k, s_j) can have a non-zero distance only if $m_j(C) = 0$ and $m_k(C) > 1$. That is, if the distance from s_k to s_j for a configuration C is non-zero, no agent has the state s_j and two or more agents necessarily have s_k. Then, the value $d_{k,j}(C)$ means how many interactions are necessary to create an agent with the state s_j from an agent having s_k in C. The distance $d_{k,j}(C)$ is obtained by multiplying the surplus number of agents having the state s_k by such the necessary number of interactions.

The following lemmas show that if the total distance of some state becomes zero, it remains zero in any following execution.

Lemma 7. *If $m_k(C) > 0$ ($0 \leq k < n$) holds in a configuration C, then $m_k(C') > 0$ in C' holds for any configuration C' such that $C \xrightarrow{*} C'$.*

Proof. Any active interaction of the Protocol 1 reduces the number of $m_k(C)$ by exactly one for some k. In addition, to enable the interaction that reducing $m_k(C)$, it is necessary that at least two agents have state s_k. So no interaction reduces $m_k(C)$ from 1 to 0. This implies that $m_k(C)$ never becomes zero after it becomes more than zero. □

Lemma 8. *Let E be any unfair execution of Protocol 1 (i.e., $E \in \mathcal{E}_U(1)$). If $m_j(C) = 0$ holds for some j in a configuration C which appears in E, a configuration C' such that $m_j(C') > 0$ is reachable from C in E.*

Proof. Clearly, in C', the total distance of the state s_j is zero. In addition, for any configuration C'', if $m_j(C'') = 0$, two or more agents have the same state in C'', which implies that as long as no agent has the state s_j, some active interaction eventually occurs (notice that it holds even under the unfairness assumption). Thus, it is sufficient to show that any active interaction decreases the total distance of s_j by one. Let

two agents having state s_k interact at a configuration C_1, and C_2 be the resultant configuration of the interaction. Then, except for $i = k, k+1$, $d_{i,j}(C_1) = d_{i,j}(C_2)$ necessarily holds because $m_i(C_1) = m_i(C_2)$ holds for any i other than k and $k+1$. In addition, by the transition, the number of agents with s_k decreases by one, and the number of agents with s_{k+1} increases by one. Thus, by simple calculation, we can obtain $d_{k,j}(C_1) + d_{k+1,j}(C_1) = d_{k,j}(C_2) + d_{k+1,j}(C_2) + 1$. This implies that $d_j(C_1) = d_j(C_2) + 1$ holds, which means any active interaction decreases the total distance of s_j by one and eventually $m_j(C)$ will increase from 0 to 1. □

By the above two lemmas, we know that if $m_j(C) = 0$, eventually there exists a configuration C' which is reachable from C such that $m_j(C') > 0$. And there is no execution can reduce $m_j(C) > 0$ to 0. Because the number of agents is the same as the number of distinct states, so we can show the following corollary.

Corollary 1. *For any execution $E = C_0, r_0, C_1, r_1, C_2, \cdots \in \mathcal{E}_U(7)$, there exists i such that $\#_0(C_j) = 0$ holds for any $j \geq i$.*

Corollary 1 and Lemma 6 directly imply the correctness of the protocol, and we can get the following theorem.

Theorem 2. *Protocol 1 is a self-stabilizing leader election one-way protocol working correctly under the unfairness assumption, and an arbitrary configuration converges to a legitimate configuration in $\Theta(n^2)$ active interactions.*

5 No Single Protocol for Complete Graphs with Difference Sizes

In Section 4, we give a protocol using n states to solve the self-stabilizing leader election in a complete graph of size n. In this section, we will show that there does not exist any single protocol to solve the self-stabilizing leader election in complete graphs with different sizes.

Theorem 3. *Letting $\mathcal{B}$ be a protocol which can solve the self-stabilizing leader election in complete graphs with size n, then $\mathcal{B}$ cannot work correctly in complete graphs with size $n-1$.*

Proof. Consider the legitimate configuration C of a complete graph with size n. Since a new leader will not be created, so the subconfiguration D which obtained by masking the leader state in C where $|D| = n-1$, cannot generate the leader state. So consider an initial configuration $C' = D - \{\perp\}$ (a configuration of a complete graph with size $n-1$ and whose entries are the non-$\perp$ entries in D), from which the leader state will not be generated when using protocol $\mathcal{B}$. Noting that C' is an initial configuration of a complete graph with size $n-1$, and from such the initial configuration, the legitimate configuration will never be reached. Hence, $\mathcal{B}$ cannot elect a leader correctly in complete graphs with size $n-1$. □

The above theorem also shows that even the upper bound for n is not sufficient knowledge to realize any SS-LE protocol.

6 Conclusion

In this paper, we have shown the necessary and sufficient conditions to the solvability of the SS-LE in population protocols having no oracles and of complete interaction graphs. The conditions are characterized by local memory space and fairness assumptions. To prove the impossibility under global fairness, we introduce a new proof technique using closed sets.

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing 18(4), 235–253 (2006)
2. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing population protocols. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 103–117. Springer, Heidelberg (2006)
3. Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 63–74. Springer, Heidelberg (2005)
4. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: Proc. 25th Annual ACM Symposium on Principles of Distributed Computing, pp. 292–299 (2006)
5. Angluin, D., Aspnes, J., Eisenstat, D.: A simple protocol for fast robust approximate majority. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 20–32. Springer, Heidelberg (2007)
6. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20(4), 279–304 (2007)
7. Aspnes, J., Ruppert, E.: An introduction to population protocols. Bulletin of the EATCS 93, 98–117 (2007)
8. Beauquier, J., Clement, J., Messika, S., Rosaz, L., Rozoy, B.: Self-stabilizing counting in mobil sensor networks. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 63–76. Springer, Heidelberg (2007)
9. Canepa, D., Potop-Butucaru, M.G.: Stabilizing leader election in population protocols (2007) (unpublished)
10. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: When birds die: Making population protocols fault-tolerant. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 51–66. Springer, Heidelberg (2006)
11. Fischer, M.J., Jiang, H.: Self-stabilizing leader election in networks of finite-state anonymous agent. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 395–409. Springer, Heidelberg (2006)

Characterizing Topological Assumptions of Distributed Algorithms in Dynamic Networks*

Arnaud Casteigts[1], Serge Chaumette[2], and Afonso Ferreira[3,**]

[1] SITE, University of Ottawa
casteig@site.uottawa.ca
[2] LaBRI, Université de Bordeaux
serge.chaumette@labri.fr
[3] CNRS - MASCOTTE Project, INRIA Sophia Antipolis
afonso.ferreira@sophia.inria.fr

Abstract. Besides the complexity in time or in number of messages, a common approach for analyzing distributed algorithms is to look at their assumptions on the underlying network. This paper focuses on the study of such assumptions in dynamic networks, where the connectivity is expected to change, predictably or not, during the execution. Our main contribution is a theoretical framework dedicated to such analysis. By combining several existing components (local computations, graph relabellings, and evolving graphs), this framework allows to express detailed properties on the network dynamics and to prove that a given property is necessary, or sufficient, for the success of an algorithm. Consequences of this work include (i) the possibility to compare distributed algorithms on the basis of their topological requirements, (ii) the elaboration of a formal classification of dynamic networks with respect to these properties, and (iii) the possibility to check automatically whether a network trace belongs to one of the classes, and consequently to know which algorithm should run on it.

Keywords: Dynamic networks, distributed algorithms, evolving graphs, local interactions, topological assumptions.

1 Introduction

The past decade has seen a considerable research effort devoted to the design of distributed algorithms and protocols targeting dynamic network topologies. It appears, however, that most of the assumptions considered when examining algorithm requirements still relate to static properties such as the *size*, *density*, or *geometry* of the target network. Assumptions that really relate to the network dynamics, when any, are generally stated using non-formal expressions, such as "*nodes are expected to move slowly enough to..*", or "*nodes cannot leave the network for a long time*", or by assuming a given *mobility model* (whose concrete

* Partially supported by A.N.R. grant No ANR-05-SSIA-0002-01.

** A.Ferreira is currently on leave as Head of Science Operations at the COST Office, Brussels, BE. http://www.cost.esf.org.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 126–140, 2010.

topological implications remain unclear). The dynamic properties highlighted in such a way make the fair comparison of algorithm requirements rather complicated and often ambiguous.

Our work aims at providing general formalisms and methods for studying fundamental properties of dynamic networks, and more particularly their impact on distributed systems. As an illustrative example, let us consider the broadcasting of an information within the dynamic network depicted by Figure 1. The possibility to complete the broadcast in this scenario clearly depends on which node is the initial emitter: a and b may succeed, while c cannot. Why? How can we formulate this intuitive property that the topology evolution must have with regard to the emitter and other nodes? How can we formally prove it as a necessary condition to obtain broadcast completion? While rather simple, such a characterization might be difficult to obtain with usual graph formalisms and computation models.

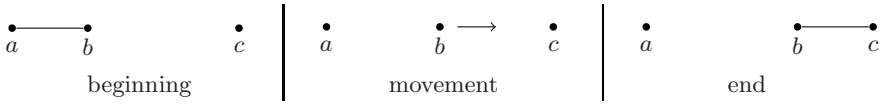


Fig. 1. A basic dynamic scenario, where a node (b) moves during the execution

This paper introduces a theoretical framework dedicated to such kind of analyses. This framework is intended to serve as a general basis for studying fundamental properties of distributed algorithms in dynamic networks. Contrary to the work in [AAD+06], where the authors first make a strong topological assumption (all pairs of nodes repeatedly meet during the execution) and then characterize the solvable problems in this context, we consider the exact opposite approach by studying, for a given solution (*i.e.*, algorithm), the *necessary* and/or *sufficient* conditions it requires on the topology. To the best of our knowledge, this is the first attempt in such a direction.

The strength of the proposed framework lies in its basic components, which are an appropriate combinatorial model to represent dynamic topologies (*evolving graphs* [Fer04]), and a very high-level interaction model to describe distributed operations (*local computations*, with the associated formalism of *graph relabellings* [LMS99]). The next section is devoted to the presentation of these existing components. In Section 3 we combine them to set up the new analysis framework. This framework is then applied in Section 4 to the analysis of three basic algorithms (one propagation and two enumeration algorithms), whose intuitively apparent properties are here formally characterized. Based on the analysis results, Section 5 shows how algorithms can be compared on the basis of their topological requirements, and reciprocally how dynamic networks can be classified according to the algorithms they support. Finally, we discuss the possibility to check automatically the inclusion of a given network trace to one of the classes. Section 6 concludes with some avenues for further research.

2 Related Work

We describe here the formalisms and theoretical tools that compose the proposed analysis framework: *Local Computations* to abstract the communication model, *Graph Relabelling Systems* to describe local computation algorithms, and *Evolving Graphs* to express properties on dynamic topologies. Their comprehension is required to ensure a clear understanding of the following sections, where they are combined together.

2.1 Abstracting Communications through Local Computations and Graph Relabellings

Distributed algorithms can be expressed using a variety of communication models (*e.g.* mailbox, shared memory, and message passing). Whereas a vast majority of algorithms is designed in one of these models (predominantly the message passing model), the very fact that one of them is chosen implies that the obtained results (*e.g.* positive or negative characterizations and associated proofs) are limited to the scope of this model. This problem of diversity among formalisms and results, already pointed out twenty years ago in [Lyn89], led researchers to consider higher abstractions when studying fundamental properties of distributed systems.

Local computations and *Graph relabellings* were jointly proposed in this perspective in [LMS99]. These theoretical tools allow to represent a distributed algorithm as a set of local interaction rules that are independent from the effective communications. Within the formalism of graph relabellings, the network is represented by graph whose vertices and edges are associated with labels that represent the algorithmic state of the corresponding nodes and links. An interaction rule is then defined as a transition pattern $(preconditions, actions)$, where *preconditions* and *actions* relate to the label values. Since these interactions are local, each transition pattern must involve a limited and connected subset of vertices and edges. Figure 2 shows different scopes for the transition patterns, which are not necessarily the same for *preconditions* and *actions*.

More formally, let the network topology be represented by a finite undirected loopless graph $G = (V_G, E_G)$, with V_G representing the set of nodes and E_G

Fig. 2. Different powers of local computations; the scope of *preconditions* is depicted in *white (on left sides)*, while the scope of *actions* is depicted in *black (on right sides)*. The *dashed elements* represent entities (vertices or edges) that are considered by preconditions but remain unaffected by actions. The reader is referred to [CMZ06] for a comparative study of some of these models.

representing the set of communication links between them. Two vertices u and v are said *neighbors* if and only if they share a common edge $\{u, v\}$ in E_G. Let $\lambda : V_G \cup E_G \rightarrow \mathcal{L}^*$ be a mapping that associates every vertex and edge from G with one or several labels from an alphabet $\mathcal{L}$ (which denotes all the possible states these elements can take). The state of a given vertex v, *resp.* edge e, at a given time t is thus denoted by $\lambda_t(v)$, *resp.* $\lambda_t(e)$. The whole *labelled graph* is represented by the pair (G, λ), noted $\mathbf{G}$.

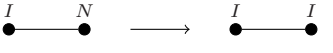
According to [LMS99], a complete algorithm can be given by a triplet $\{\mathcal{L}, \mathcal{I}, P\}$, where $\mathcal{I}$ is the set of initial states, and P is a set of *relabelling rules* (transition patterns) representing the distributed interactions. The Algorithm 1 below ($\mathcal{A}_1$ for short), gives the example of a one-rule algorithm that represents the general broadcasting scheme discussed in the introduction. We assume here that the label I (*resp.* N) stands for the state `informed` (*resp.* `non-informed`). Propagating the information thus consists in repeating this single rule, starting from the emitter vertex, until all vertices are labelled I.[1]

Algorithm 1. A propagation algorithm coded by a single relabelling rule (r_1).

initial states: $\{I, N\}$ (I for the initial emitter, N for all other vertices)
alphabet: $\{I, N\}$

preconditions(r_1): $\lambda(v_0) = I \wedge \lambda(v_1) = N$
actions(r_1): $\lambda(v_1) := I$

graphical notation :



Remark 1. Although the three algorithm examples provided in this paper consider pairwise interactions (and more specifically the model depicted on Figure 2(c)), the concepts and discussions developed in Sections 3 and 5 are not dedicated to it. Note that models such as those of Fig. 2(b) and 2(b) reflect well a wireless computing environment where nodes update their states according to those of their neighbors.

Regarding the organization of collaborations between nodes, it is important to note that the algorithm specification does not stipulate how the nodes must collaborate, *i.e.,* the way they select each other to perform a common computation step. From the abstraction level of local computations, this underlying synchronization is seen as an implementation choice, which implies that local computation algorithms may not be deterministic at this level. As discussed later on, characterizing *sufficient* conditions will require additional assumptions on this underlying layer (which is not the case for *necessary* conditions).

2.2 Expressing Dynamic Network Properties Using Evolving Graphs

In a different context, *evolving graphs* [Fer04] have been proposed as a combinatorial model for dynamic networks. The initial purpose of this model was to

[1] Detecting such a final state is not part of the given algorithm. The reader interested in termination detection as a distributed problem is referred to [GMMS02].

provide a suitable representation of *fixed schedule dynamic networks* (FSDNs), in order to compute optimized communication schemes such as shortest, fastest and foremost paths. In such a context, the evolution of the network was known beforehand. In the present work, we propose to use evolving graphs for a different purpose, namely to express topological properties in dynamic networks. It is important to keep in mind that the analyzed algorithms are never supposed to know the evolution of the network ahead of time.

An evolving graph is a structure in which the changing connectivity of a dynamic network is recorded (see Figure 3). More formally, let $\mathcal{S}_\mathbb{T} = t_0, t_1, ..., t_n$ be a sequence of dates in $\mathbb{T}$ (usually $\mathbb{R}^+$). Except for t_0 and t_n, all these dates correspond to a topological event that modifies the network. Let $\mathcal{S}_G = G_0, G_1, ..., G_{n-1}$ be the corresponding sequence of graphs, with each G_i the graph corresponding to the period $[t_i, t_{i+1}[$. Finally, let us denote by G (alone) the union graph of all G_i, called the *underlying graph*. Then the triplet $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_\mathbb{T})$ is the corresponding *evolving graph*. As shown in Figure 3, this graph can be represented by the underlying graph G whose edges and vertices (only edges here) are associated with their presence interval indices. Henceforth, we will use the notations $V_\mathcal{G}$ and $E_\mathcal{G}$ to denote $V(G)$ and $E(G)$, the sets of all vertices and edges that exist at some point of the network life. Note that whereas used as *undirected* in this paper, evolving graphs were initially introduced as *directed*, and considered also bandwidth restrictions on edges, which is not used here.

Further definitions on evolving graphs (given an evolving graph $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_\mathbb{T})$).
Predecessor of a date: for any date d in $\mathbb{T}_{[t_i, t_{i+1}[}$, with $t_i, t_{i+1} \in \mathcal{S}_\mathbb{T}$, we say that t_i is the *predecessor* of d in $\mathcal{S}_\mathbb{T}$, and we note $pred(d) = t_i$.
Journey: given a sequence of couples $\mathcal{J} = \{(e_1, \sigma_1), ..., (e_i, \sigma_i), ..., (e_k, \sigma_k)\}$ composed of edges from $E_\mathcal{G}$ and dates from the continuous domain $\mathbb{T}$, $\mathcal{J}$ is called a *journey* if and only if $\sigma_1, \sigma_2, ..., \sigma_k$ is non-decreasing and for all i in $1..k$, $e_i \in E(G_{pred(\sigma_i)})$, that is, the edge e_i exists at time σ_i. Less formally, a journey

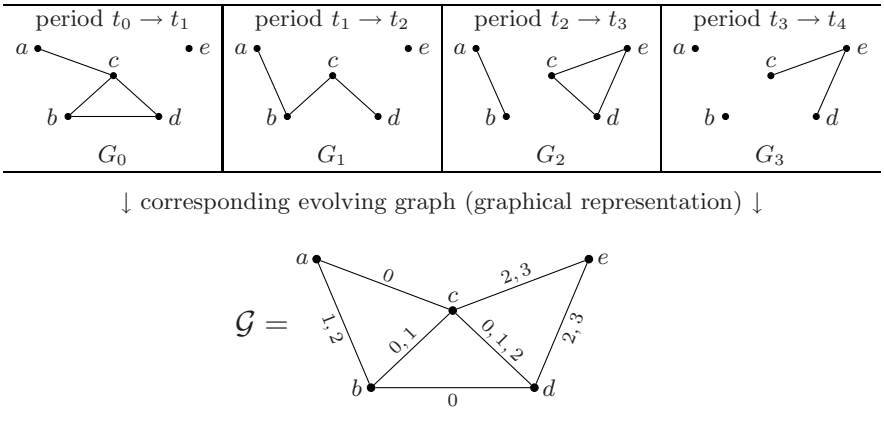


Fig. 3. Example of an evolving graph covering a period of time from t_0 to t_4

can be thought of as a path *over-time* from one vertex to another. A journey from a vertex u to a vertex v is noted $\mathcal{J}_{(u,v)}$.
Discrete Journey: a *discrete journey* is a journey so that every date of the sequence $\sigma_1, ..., \sigma_k$ is in $\mathcal{S}_{\mathbb{T}}$, instead of $\mathbb{T}$. It allows to represent in a single entity all the possible journeys occurring on the same sequence of edges during the same sequence of intervals. This also allows to consider such entities as *subgraphs* of the evolving graph $\mathcal{G}$, and to note $\mathcal{J} \subseteq \mathcal{G}$. The point is that every normal journey $\{(e_1, \sigma_1), ..., (e_i, \sigma_i), ..., (e_k, \sigma_k)\}$ can be associated with a discrete journey $\{(e_1, pred(\sigma_1)), ..., (e_i, pred(\sigma_i)), ..., (e_k, pred(\sigma_k))\} \subseteq \mathcal{G}$, and every discrete journey implies an infinity of normal journeys for the corresponding edges and intervals.
Strictness of a discrete journey: a discrete journey is said *strict*, noted $\mathcal{J}_{strict}$, if its sequence of dates $\sigma_1, \sigma_2, ..., \sigma_k$ is strictly increasing.

To give a few examples on the graph of Figure 3,
-$J_{(a,e)}$=$\{(ac, \sigma_1 \in [t_0, t_1[), (ce, \sigma_2 \in [t_2, t_3[)\}$ is a *normal* journey from a to e ;
-$J_{strict(a,e)}$=$\{(ac, 0), (ce, 2)\}$ is a *discrete* (and strict) journey from a to e ;
-$J_{(a,e)}$=$\{(ac, 0), (cd, 0), (de, 3)\}$ is a discrete (non-strict) journey from a to e ;
-$J_{strict(a,e)}$=$\{(ac, 0), (cd, 1), (de, 3)\}$ is a discrete (and strict) journey from a to e.

Note that journeys are naturally oriented, in the sense that a journey from one vertex to another does not imply the existence of a journey in the reverse direction (*e.g.* from e to a). From this point on, unless said explicitly, we will only consider discrete journeys, and denote them by the sole term *journey*.

3 The Proposed Analysis Framework

As a recall of the previous section, the algorithmic state of the network is given by a labelling on the corresponding graph G, then noted $\mathbf{G}$. As another recall, we denote by G_i the graph covering the period $[t_i, t_{i+1}[$ in the evolving graph $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$, with $G_i \in \mathcal{S}_G$ and $t_i, t_{i+1} \in \mathcal{S}_{\mathcal{T}}$. Note that the notation G was used here with two different meanings: the first as the generic letter to represent the network, the second to denote the *underlying graph* of $\mathcal{G}$. Both notations are kept in the following, while preventing the text from ambiguities.

3.1 Putting the Pieces Together: Relabellings over Evolving Graphs

For an evolving graph $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$ and a given date index $i \mid t_i \in \mathcal{S}_{\mathbb{T}}$, we denote by $\mathbf{G}_i$ the labelled graph $(G_i, \lambda_{t_i+\epsilon})$ representing the network state just after the topological event of date t_i, and by $\mathbf{G}_{i[}$ the labelled graph $(G_{i-1}, \lambda_{t_i-\epsilon})$ representing the network state just before it. We note,

$$Event_{t_i}(\mathbf{G}_{i[}) = \mathbf{G}_i \ .$$

A number of distributed operations can occur between two consecutive events. Hence, for a given algorithm $\mathcal{A}$ and two consecutive dates $t_i, t_{i+1} \in \mathcal{S}_{\mathcal{T}}$, we

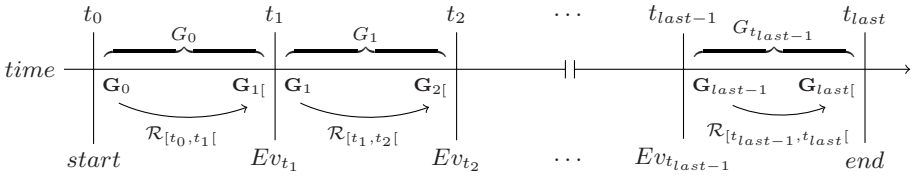


Fig. 4. Graph Relabellings and Evolving Graphs - Combined formalism

denote by $\mathcal{R}_{\mathcal{A}_{[t_i,t_{i+1}[}}$ the relabelling sequence induced by $\mathcal{A}$ on the graph G_i during the period $[t_i, t_{i+1}[$, and have,

$$\mathcal{R}_{\mathcal{A}_{[t_i,t_{i+1}[}}(\mathbf{G}_i) \;=\; \mathbf{G}_{i+1[}\ .$$

For the sake of simplicity, we authorize the notation $r_i(u,v) \in \mathcal{R}_{\mathcal{A}_{[t,t'[}}$ to denote the fact that a rule r_i is applied on the edge (u,v) during $[t,t'[$. A complete execution sequence from t_0 to t_{last} is given by an alternated sequence of relabelling steps and topological events, noted,

$$X{=}\mathcal{R}_{\mathcal{A}_{[t_{last-1},t_{last}[}} \circ Event_{t_{last-1}} \circ .. \circ Event_{t_i} \circ \mathcal{R}_{\mathcal{A}_{[t_{i-1},t_i[}} \circ .. \circ Event_{t_1} \circ \mathcal{R}_{\mathcal{A}_{[t_0,t_1[}}(\mathbf{G}_0)$$

The combined formalism is summed up on Figure 4. As mentioned at the end of Section 2.1, the execution of a local computation algorithm is not necessarily deterministic, and may depend on the way nodes select one another at a lower level. Hence, we denote by $\mathcal{X}_{\mathcal{A}/\mathcal{G}}$ the set of all possible execution sequences of an algorithm $\mathcal{A}$ over an evolving graph $\mathcal{G}$.

3.2 Characterizing the Topological Assumptions of an Algorithm

Below are some proposed methods and additional concepts to characterize the requirement of an algorithm in terms of topology dynamics. More precisely, we use the new combined formalism to define the notions of topology-related *necessary* and *sufficient* conditions, and discuss how they can be proved.

Objectives of an algorithm. Given an algorithm $\mathcal{A}$ and a labelled graph $\mathbf{G}$, the state one desires to reach can be given by a logic formula $\mathcal{P}$ on the labels of vertices and/or edges. In the case of the propagation scheme (Algorithm 1 Section 2.1), this could be that all nodes are informed,

$$\mathcal{P}_1(\mathbf{G}) = \forall v \in V(\mathbf{G}), \lambda(v) = I\ ,$$

Now, if the objective (noted $\mathcal{O}$) is to *reach* such state at some point, then it can be simply expressed as $\mathcal{P}$ to be satisfied on the very last labelled graph of $\mathcal{G}$ (*e.g.* $\mathcal{O}_{\mathcal{A}_1} = \mathcal{P}_1(\mathbf{G}_{last})$ in the example). Whereas not covered in the examples, one could also consider algorithms whose objectives are to *maintain* a state (*e.g. self-stabilizing* algorithms), and express it for example as $\mathcal{O}_{\mathcal{A}} = \forall G_i \in S_G, \mathcal{P}(\mathbf{G}_{i+1[})$.

Necessary conditions. Given an algorithm $\mathcal{A}$, its objective $\mathcal{O}_\mathcal{A}$, an evolving graph $\mathcal{G}$ and an evolving graph property $\mathcal{C}_\mathcal{N}$. The property $\mathcal{C}_\mathcal{N}$ is a *(topology-related) necessary* condition for $\mathcal{O}_\mathcal{A}$ if and only if

$$\forall \mathcal{G}, \neg \mathcal{C}_\mathcal{N}(\mathcal{G}) \implies \neg \mathcal{O}_\mathcal{A}$$

Proving this result comes to prove that $\forall \mathcal{G}, \neg\, \mathcal{C}_\mathcal{N}(\mathcal{G}) \implies \nexists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}} \mid \mathcal{P}(\mathbf{G}_{last})$.

Sufficient conditions. Symmetrically, an evolving graph property $\mathcal{C}_\mathcal{S}$ is a *(topology-related) sufficient* condition for $\mathcal{A}$ if and only if

$$\forall \mathcal{G}, \mathcal{C}_\mathcal{S}(\mathcal{G}) \implies \mathcal{O}_\mathcal{A}$$

Proving this result comes to prove that $\forall \mathcal{G}, \mathcal{C}_\mathcal{S}(\mathcal{G}) \implies \forall X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \mathcal{P}(\mathbf{G}_{last})$.

Discussion. No topology of any kind can guarantee, alone, that the nodes will effectively communicate and collaborate with each other. Hence, the characterization of any sufficient condition necessarily requires to make additional assumptions on the collaboration of nodes. We propose below a generic such assumption for the pairwise interaction model (depicted on Figure 2(c)). This assumption may or may not be considered as realistic depending on the expected rate of topological changes.

Progression Hypothesis 1 *(PH_1). For every given time interval $[t_i, t_{i+1}[$, with t_i in $\mathcal{S}_\mathbb{T}^{\setminus\{t_{last}\}}$, every vertex will be able to apply at least one relabelling rule with each of its neighbors, provided the rule preconditions are* already *satisfied at time t_i (and still satisfied at the time the rule is applied).*

4 First Applications of the Proposed Framework

This section illustrates the proposed framework by the analysis of three basic algorithms, namely the propagation algorithm previously given, and two enumeration algorithms (one centralized, the other decentralized). The results obtained here are used in the next section to highlight some implications of this work.

4.1 Analysis of the Propagation Algorithm

We want to prove that the existence of a journey (*resp.* strict journey) between the emitter and every other node is a necessary (*resp.* sufficient) condition to achieve $\mathcal{O}_{\mathcal{A}_1}$ (complete the propagation). The point here is to show how these intuitive conditions can be *formally* established.

Condition 1. $\forall v \in V_\mathcal{G}^{\setminus\{emitter\}}, \exists \mathcal{J}_{(emitter,v)} \subseteq \mathcal{G}$
(It exists a journey between the emitter and every other vertex).

Lemma 1. $\forall v \in V_\mathcal{G} \mid \lambda_{t_0}(v) = N, \forall \sigma \in \mathbb{T}_{[t_0,t_{last}[}, \lambda_\sigma(v) = I \implies \exists u \in V_\mathcal{G}^{\setminus\{v\}}, \sigma' \in \mathbb{T}_{[t_0,\sigma[} \mid \lambda_{\sigma'}(u) = I, \exists \mathcal{J}_{(u,v)} \subseteq \mathcal{G}$
(If a non-emitter vertex has the information at some point, it implies the existence of an incoming journey from a vertex that had the information before)

Proof. $\forall v \in V_{\mathcal{G}} \mid \lambda_{t_0}(v) = N, \forall \sigma \in \mathbb{T}_{[t_0,t_{last}[}, (\lambda_\sigma(v) = I \implies \exists v' \in V_{\mathcal{G}}^{\setminus v} \mid r_{1(v',v)} \in \mathcal{R}_{\mathcal{A}_1[t_0,\sigma[})$ (If a non-emitter vertex has the information at some point, then it has applied rule r_1 with another vertex)
$\implies \exists v' \in V_{\mathcal{G}}^{\setminus v}, \sigma' \in \mathbb{T}_{[t_0,\sigma[} \mid \lambda_{\sigma'}(v') = I, (v', v) \in E(G_{pred(\sigma')})$
(An edge existed at a previous date between this vertex and a vertex labelled I)
By repetition, $\implies \exists v'' \in V_{\mathcal{G}}^{\setminus v}, \sigma'' \in \mathbb{T}_{[t_0,\sigma[} \mid \lambda_{\sigma''}(v'') = I, \exists \mathcal{J}_{(v'',v)} \subseteq \mathcal{G}$
(A journey existed from a node that had the information to the considered node) □

Proposition 1. *Condition 1 ($\mathcal{C}_1$) is a necessary condition on $\mathcal{G}$ to allow Algorithm 1 ($\mathcal{A}_1$) to reach its objective $\mathcal{O}_{\mathcal{A}_1}$.*

Proof. (using Lemma 1). Following from Lemma 1 and the initial states (I for the emitter, N for all other vertices), we have $\mathcal{O}_{\mathcal{A}_1} \implies \mathcal{C}_1$, and then $\neg\mathcal{C}_1 \implies \neg\mathcal{O}_{\mathcal{A}_1}$ □

Condition 2. $\forall v \in V_{\mathcal{G}}^{\setminus\{emitter\}}, \exists \mathcal{J}_{strict(emitter,v)} \subseteq \mathcal{G}$

Proposition 2. *Assuming the progression hypothesis (PH_1, defined in the previous section), Condition 2 ($\mathcal{C}_2$) is sufficient on $\mathcal{G}$ to guarantee that $\mathcal{A}_1$ will reach $\mathcal{O}_{\mathcal{A}_1}$.*

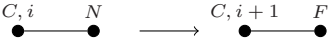
Proof. (1): By PH_1, $\forall t_i \in \mathcal{S}_{\mathbb{T}}^{\setminus(t_{last})}, \forall (u,u') \in E(G_i), (\lambda_{t_i}(u) = I \implies \lambda_{t_{i+1}}(u') = I)$
By iteration on (1): $\forall u, v \in V_{\mathcal{G}}, (\exists \mathcal{J}_{strict(u,v)} \subseteq \mathcal{G}) \implies (\lambda_{t_0}(u){=}I \implies \lambda_{t_{last}}(v){=}I)$
Now, because $\lambda_{t_0}(emitter) = I$, we have $\mathcal{C}_2(\mathcal{G}) \implies \forall X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \mathcal{P}_1(\mathbf{G}_{last})$ □

4.2 Analysis of a Centralized Enumeration Algorithm

Like the propagation algorithm, the distributed algorithm presented below assumes that one distinguished vertex is given a different initial state. This vertex, called the *counter*, is in charge of counting all the vertices it meets during the execution (its successive neighbors in the changing topology). Hence, the counter vertex has two labels (C, i), meaning that it is the counter (C), and that it has already counted i vertices (initially 1, *i.e.,* itself). The other vertices are labelled either F or N, depending on whether they have already been counted or not, respectively. The counting rule is given by r_1 in Algorithm 2, below.

Algorithm 2. Enumeration algorithm with a pre-selected counter.

initial states: $\{(C,1), N\}$ ($(C,1)$ for the counter, N for all other vertices)
alphabet: $\{C, N, F, \mathbb{N}^*\}$
rule r_1:



Objective of the algorithm. Under the assumption of a fixed number of vertices, the algorithm reaches the desired state when all vertices are counted, which corresponds to the fact that no more vertices are labelled N:

$$\mathcal{P}_2 = \forall v \in V(\mathbf{G}), \lambda(v) \neq N$$

The objective of Algorithm 2 is then to satisfy this property at the end of the execution ($\mathcal{O}_{\mathcal{A}_2} = \mathcal{P}_2(\mathbf{G}_{last})$). We want to prove here that the existence of an edge at some point of the execution between the *counter* node and every other node is a necessary and sufficient condition.

Condition 3. $\forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \exists t_i \in \mathcal{S}_{\mathbb{T}} \mid (counter, v) \in E(G_i)$, *or equivalently with the notion of underlying graph,* $\forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, (counter, v) \in E_{\mathcal{G}}$

Proposition 3. *For a given evolving graph $\mathcal{G}$ representing the topological evolutions that take place during the execution of $\mathcal{A}_2$, Condition 3 ($\mathcal{C}_3$) is a* necessary *condition on $\mathcal{G}$ to allow $\mathcal{A}_2$ to reach its objective $\mathcal{O}_{\mathcal{A}_2}$.*

Proof. $\neg\mathcal{C}_3(\mathcal{G}) \implies \exists v \in V_{\mathcal{G}}^{\setminus\{counter\}} \mid (counter, v) \notin E(G)$
$\implies \exists v \in V_{\mathcal{G}}^{\setminus\{counter\}} \mid \forall t_i \in \mathcal{S}_{\mathbb{T}}^{\setminus\{t_{last}\}}, r_1(counter, v) \notin \mathcal{R}_{\mathcal{A}_2[t_i, t_{i+1}[}$
$\implies \exists v \in V_{\mathcal{G}}^{\setminus\{counter\}} \mid \forall X \in \mathcal{X}_{\mathcal{A}_2/\mathcal{G}}, \lambda_{t_{last}}(v) = N$
$\implies \nexists X \in \mathcal{X}_{\mathcal{A}_2/\mathcal{G}} \mid \mathcal{P}_2(\mathbf{G}_{last}) \implies \neg\mathcal{O}_{\mathcal{A}_2}$ □

Proposition 4. *Assuming the progression hypothesis (PH_1), $\mathcal{C}_3$ is also a sufficient condition on $\mathcal{G}$ to guarantee that $\mathcal{A}_2$ will reach its objective $\mathcal{O}_{\mathcal{A}_2}$.*

Proof. $\mathcal{C}_3(\mathcal{G}) \implies \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \exists t_i \in \mathcal{S}_{\mathbb{T}} \mid (counter, v) \in E(G_i)$
by PH_1, $\implies \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \exists t_i \in \mathcal{S}_{\mathbb{T}} \mid r_1(counter, v) \in \mathcal{R}_{\mathcal{A}_2[t_i, t_{i+1}[}$
$\implies \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \lambda_{t_{last}}(v) \neq N$
$\implies \forall X \in \mathcal{X}_{\mathcal{A}_2/\mathcal{G}}, \mathcal{P}_2(\mathbf{G}_{last}) \implies \mathcal{O}_{\mathcal{A}_2}$ □

4.3 Analysis of a Decentralized Enumeration Algorithm

Contrary to the previous algorithm, Algorithm 3 below does not require a distinguished initial state for any vertex. Indeed, all vertices are initialized with the same labels $(C, 1)$, meaning that they are all initially counters that have already included themselves into the count. Then, depending on the topological evolutions, the counters opportunistically merge by pairs (rule r_1) in Algorithm $\mathcal{A}_3$. In the optimal case, at the end of the execution, only one node remains labelled C and its second label gives the total number of vertices in the graph. A similar counting principle has been used in [AAD+06] to monitor a flock of birds for fever, with the role of counters being played by sensors that have measured a high temperature level.

Algorithm 3. Decentralized enumeration algorithm.

initial states: $\{(C, 1)\}$ (for all vertices); *alphabet:* $\{C, F, \mathbb{N}^*\}$
rule r_1:

Objective of the algorithm. Under the assumption of a fixed number of vertices, this algorithm reaches the desired state when exactly one vertex remains labelled C:

$$\mathcal{P}_3 = \exists u \in V_{\mathcal{G}} \mid \forall v \in V_{\mathcal{G}}^{\setminus\{u\}}, \lambda(u) = C, \lambda(v) \neq C, \text{ and } \mathcal{O}_{\mathcal{A}_3} = \mathcal{P}_3(\mathbf{G}_{last})$$

For the sake of simplicity, we introduce one additional definition: the *destination set* of a vertex v in an evolving graph $\mathcal{G}$ is the set of all the vertices that can be reached from v by a journey, noted $\mathcal{D}est_{\mathcal{G}}(v)$. Note that $v \in \mathcal{D}est_{\mathcal{G}}(v)$ through an empty journey. We want to prove here that the existence of a journey from every vertex to at least one common destination vertex is a necessary condition for this algorithm.

Condition 4. $\exists v \in V_{\mathcal{G}} \mid \forall u \in V_{\mathcal{G}}, v \in \mathcal{D}est_{\mathcal{G}}(u)$

Lemma 2. $\forall u \in V_{\mathcal{G}} \mid \lambda_{t_i}(u) = C, \exists u' \in \mathcal{D}est_{\mathcal{G}}(u) \mid \lambda_{t_{j\geq i}}(u') = C$
(Whatever the C-labelled vertex considered at some point, there will be at a later point of the execution at least one vertex labelled C among its destination vertices)

Proof. (by contradiction). The application of r_1 is the only operation that can suppress a counter, while preserving the other counter in the pair. If Lemma 2 was false, then it would imply either that both counters have been discarded by r_1 at some point, or that the relabelling sequence has occurred from a C-labelled vertex towards a vertex that is outside of its destination set. Both are impossible. □

Proposition 5. *Condition 4 ($\mathcal{C}_4$) is* necessary *for $\mathcal{A}_3$ to reach its objective $\mathcal{O}_{\mathcal{A}_3}$.*

Proof. (using Lemma 2). $\neg\mathcal{C}_4(\mathcal{G}) \implies \nexists v \in V_{\mathcal{G}} \mid \forall u \in V_{\mathcal{G}}, v \in \mathcal{D}est_{\mathcal{G}}(u)$
(no vertices are destination for all the others).
$\implies \forall v \in V_{\mathcal{G}} \mid \lambda_{t_{last}}(v) = C, \exists u \in V_{\mathcal{G}} \mid v \notin \mathcal{D}est_{\mathcal{G}}(u)$
(Whatever the final counter, there is a vertex that could not reach it by a journey).
Now, thanks to Lemma 2, $\implies \forall v \in V_{\mathcal{G}} \mid \lambda_{t_{last}}(v) = C, \exists v' \in V_{\mathcal{G}}^{\setminus\{v\}} \mid \lambda_{t_{last}}(v') = C$
(There are at least two final counters).
$\implies \neg\mathcal{P}_3(\mathbf{G}_{last}) \implies \neg\mathcal{O}_{\mathcal{A}_3}$ □

The characterization of a sufficient condition for $\mathcal{A}_3$ is left open. We believe such a condition exists, but would be satisfied on a very few specific graphs.

5 Applications of the Analysis Results

This section presents some applications of the new framework. In particular, we show how the previously characterized conditions can be used to define evolving graph classes, some of which are included in others. This leads to a *de facto* classification of dynamic networks according to the algorithms they support. The relations between classes can be used in turn to compare algorithms on the basis of their topological requirements. Finally, we propose a method to check a given network trace for inclusion in each introduced class.

5.1 From Conditions to Graph Classes

From $\mathcal{C}_1 = \forall v \in V_{\mathcal{G}}^{\setminus\{emitter\}}, \exists \mathcal{J}_{(emitter,v)} \subseteq \mathcal{G}$, we derive two classes of evolving graphs. $\mathcal{F}_1$ is the class in which at least one vertex can reach all the others by a journey. If an evolving graph does not belong to this class, then there is no chance for $\mathcal{A}_1$ to succeed whatever the initial emitter. $\mathcal{F}_2$ is the class where every vertex can reach all the others by a journey. If an evolving graph does not belong to this class, then at least one vertex, if chosen as an initial emitter, is guaranteed to fail to inform all the others using $\mathcal{A}_1$.

From $\mathcal{C}_2 = \forall v \in V_{\mathcal{G}}^{\setminus\{emitter\}}, \exists \mathcal{J}_{strict(emitter,v)} \subseteq \mathcal{G}$, we derive two classes of evolving graphs. $\mathcal{F}_3$ is the class in which at least one vertex can reach all the others by a strict journey. If an evolving graph belongs to this class, then there is at least one vertex that could, for sure, inform all the others using $\mathcal{A}_1$ (assuming the progression hypothesis). $\mathcal{F}_4$ is the class of evolving graphs in which every vertex can reach all the others by a strict journey. If an evolving graph belongs to this class, then the success of $\mathcal{A}_1$ is guaranteed for any vertex as initial emitter (again, if the progression hypothesis is assumed).

From $\mathcal{C}_3 = \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, (counter, v) \in E_{\mathcal{G}}$, we derive two classes of graphs. $\mathcal{F}_5$ is the class of evolving graphs in which at least one vertex shares, at some point of the execution, an edge with every other vertex. If an evolving graph does not belong to this class, then there is no chance of success for $\mathcal{A}_2$, whatever the vertex chosen for counter. Here, if we assume the progression hypothesis, then $\mathcal{F}_5$ is also a class in which the success of the algorithm can be guaranteed for one specific vertex as counter. $\mathcal{F}_6$ is the class of evolving graphs in which every vertex shares an edge with every other vertex at some point of the execution. If an evolving graph does not belong to this class, then there exists at least one vertex for which, if it is chosen as the counter, the failure of $\mathcal{A}_2$ is guaranteed. Again, if we consider the progression hypothesis, then $\mathcal{F}_6$ becomes a class in which the success is guaranteed whatever the counter.

Finally, from $\mathcal{C}_4 = \exists v \in V_{\mathcal{G}} \mid \forall u \in V_{\mathcal{G}}, v \in \mathcal{D}est_{\mathcal{G}}(u)$, we derive the class $\mathcal{F}_7$, which is the class of graphs such that at least one vertex can be reached from all the others by a journey. If a graph does not belong to this class, then there is absolutely no chance of success for $\mathcal{A}_3$.

5.2 Relations between Classes

Since *all* implies *at least one*, we have: $\mathcal{F}_2 \subseteq \mathcal{F}_1$, $\mathcal{F}_4 \subseteq \mathcal{F}_3$, and $\mathcal{F}_6 \subseteq \mathcal{F}_5$. Since a strict journey is a journey, we have: $\mathcal{F}_3 \subseteq \mathcal{F}_1$, and $\mathcal{F}_4 \subseteq \mathcal{F}_2$. Since an edge is a (strict) journey, we have: $\mathcal{F}_5 \subseteq \mathcal{F}_3$, $\mathcal{F}_6 \subseteq \mathcal{F}_4$, and $\mathcal{F}_5 \subseteq \mathcal{F}_7$. Finally, the existence of a journey between all pairs of vertices ($\mathcal{F}_2$) implies that each vertex can be reached by all the others, which implies in turn that at least one vertex can be reach by all the others ($\mathcal{F}_7$). We then have: $\mathcal{F}_2 \subseteq \mathcal{F}_7$. Although we have used here a non-strict inclusion ($\subseteq$), the inclusions described above are strict ($\subset$). This can be easily proved by finding for each inclusion a graph that belongs to the parent class but is outside the child class. Figure 5 summarizes all these relations.

$\mathcal{F}_1 : \exists u \in V_\mathcal{G} \mid \forall v \in V_\mathcal{G}^{\setminus\{u\}}, \exists \mathcal{J}_{(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_2 : \forall u, v \in V_\mathcal{G}, \exists \mathcal{J}_{(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_3 : \exists u \in V_\mathcal{G} \mid \forall v \in V_\mathcal{G}^{\setminus\{u\}}, \exists \mathcal{J}_{strict(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_4 : \forall u, v \in V_\mathcal{G}, \exists \mathcal{J}_{strict(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_5 : \exists u \in V_\mathcal{G} \mid \forall v \in V_\mathcal{G}^{\setminus\{u\}}, (u,v) \in E_\mathcal{G}$
$\mathcal{F}_6 : \forall u, v \in V_\mathcal{G}, (u,v) \in E_\mathcal{G}$
$\mathcal{F}_7 : \exists u \in V_\mathcal{G} \mid \forall v \in V_\mathcal{G}^{\setminus\{u\}}, u \in \mathcal{D}est_\mathcal{G}(v)$

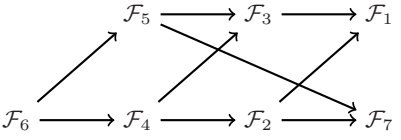


Fig. 5. A first classification of dynamic networks, based on evolving graph properties that result from the analysis of three distributed algorithms (*arrows denote inclusion*)

5.3 Comparison of Algorithms According to Topological Assumptions

Let us consider the two enumeration algorithms given in Section 4. To have any chance of success, $\mathcal{A}_2$ requires the evolving graph to be in $\mathcal{F}_5$ (and a fortunate choice of counter) or in $\mathcal{F}_6$ (for possibly any vertex as counter). On the other hand, $\mathcal{A}_3$ requires the evolving graph to be in $\mathcal{F}_7$. Now, both classes $\mathcal{F}_5$ (directly) and $\mathcal{F}_6$ (transitively) are included in $\mathcal{F}_7$. As a consequence, there are some topological scenarios (*i.e.,* $\mathcal{G} \in \mathcal{F}_7{}^{\setminus\mathcal{F}_5}$) for which $\mathcal{A}_2$ has no chance of success, while $\mathcal{A}_3$ has some. Such observation allows to claim that $\mathcal{A}_3$ is more general than $\mathcal{A}_2$ with respect to its topological requirements. Hence, two algorithms can be fairly (and formally) compared on the basis of their topological requirements. In the particular case of these two enumeration algorithms, however, the claim could be balanced by the fact that a sufficient condition is known for $\mathcal{A}_2$, while no one is known for $\mathcal{A}_3$. The choice for the right algorithm may thus depend on the target mobility context: if this context is expected to induce topological scenarios in $\mathcal{F}_5$ or $\mathcal{F}_6$, then $\mathcal{A}_2$ could be preferred, otherwise $\mathcal{A}_3$ should be considered. More generally, it is however important to realize that a large gap may exist between *necessary* and *sufficient topology-related* conditions, and other topological properties (*resp.* evolving graph classes) could offer intermediate probabilities of success, which was not investigated for the given algorithms in this initial work.

5.4 Checking Network Traces for Inclusion in the Classes

We consider here the problem of checking automatically whether a given evolving graph belongs to one of the classes listed before. While having potentially a large scope of applications, this could allow in particular to help decide which algorithm is relevant to a given mobility context, by checking how the corresponding topological traces distribute over the classes. Below is a sketch of solution for each class met so far. The point is that all solutions can rely on common *static* graph properties, provided a few transformations. The *transitive closure* of an evolving graph $\mathcal{G}$ is the graph $H = (V, A_H)$, where $A_H = \{(v_i, v_j) : \exists \mathcal{J}_{(v_i,v_j)} \subseteq \mathcal{G})\}$. A transitive closure is by nature a *directed* graph, as illustrated in Figure 6, since journeys are oriented entities. As explained in [BF03], the computation of transitive closures can be done efficiently (in $O(|V_\mathcal{G}|.|E_\mathcal{G}|.(log|\mathcal{S}_\mathbb{T}|.log|V_\mathcal{G}|))$, by

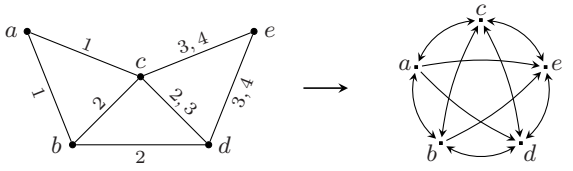


Fig. 6. Example of transitive closure of an evolving graph

building the tree of *shortest* journeys for each node in the network. We extend this notion to the case of strict journeys, with $H_{strict} = (V, A_{H_{strict}})$, where $A_{H_{strict}} = \{(v_i, v_j) : \exists \mathcal{J}_{strict(v_i,v_j)} \subseteq \mathcal{G})\}$.

Given an evolving graph $\mathcal{G}$, its underlying graph G, its transitive closure H, and the transitive closure of its *strict* journeys H_{strict}, the inclusion of $\mathcal{G}$ in each class can be checked as follows:

- $\mathcal{G} \in \mathcal{F}_1 \iff H$ contains an out-dominating set of size 1.
- $\mathcal{G} \in \mathcal{F}_2 \iff H$ is a complete graph.
- $\mathcal{G} \in \mathcal{F}_3 \iff H_{strict}$ contains an out-dominating set of size 1.
- $\mathcal{G} \in \mathcal{F}_4 \iff H_{strict}$ is a complete graph.
- $\mathcal{G} \in \mathcal{F}_5 \iff G$ contains a dominating set of size 1.
- $\mathcal{G} \in \mathcal{F}_6 \iff G$ is a complete graph.
- $\mathcal{G} \in \mathcal{F}_7 \iff H$ contains an in-dominating set of size 1.

We expect most of the future classes to be possibly checked with similar approaches. This is however not a certainty.

6 Conclusion

This paper introduced a set of tools and methods dedicated to the analysis of distributed algorithms in dynamic networks. This new framework allows to characterize assumptions that a given algorithm requires in terms of topological evolution during its execution. It was illustrated by the analysis of three basic algorithms, and the analysis results were used to highlight potential implications of this work, including the possibility to compare algorithms on the basis of their topological requirements, and a sketch of classification of dynamic networks according to the corresponding properties. The problem of checking whether a given evolving graph belongs to the introduced classes was finally discussed.

Analyzing the requirement of algorithms is not a novel approach. It appears however that no proper transposition was previously done in the context of dynamic networks, where the usual practice is to liken dynamic topologies to static graphs. This is particularly striking in the recent field of *population protocols* [AAER07], where a common assumption is that all pairs of nodes interact repeatedly. In the light of the classification shown is this paper, such scenarios actually represent a subset of the most specific class among those discussed (namely, $\mathcal{F}_6$). We think the framework proposed here could help characterize weaker assumptions for most population protocols.

The algorithms studied in this paper are simple. An interesting question for further research is whether the framework will scale to more complex algorithms, which remains unclear at this stage. We hope it could suit the study of common problems such as *electing*, *naming*, or *building spanning structures* (note that *electing* and *naming* may not have identical assumptions in a dynamic context). Another prospect is to investigate how intermediate properties could be explored between necessary and sufficient conditions, for example to guarantee a desired probability of success. Finally, as more properties are characterized and the classification grows, new insights may follow in the study of mobility models, based on checking generated traces for inclusion in the classes. Ultimately, this could answer questions like what kind of problems can be solved within a given mobility model, such as the well-known *random way point* model [BRS03], or in more realistic pedestrian and vehicular contexts.

References

[AAD+06] Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing 18(4), 235–253 (2006)

[AAER07] Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20(4), 279–304 (2007)

[BF03] Bhadra, S., Ferreira, A.: Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In: Pierre, S., Barbeau, M., Kranakis, E. (eds.) ADHOC-NOW 2003. LNCS, vol. 2865, pp. 259–270. Springer, Heidelberg (2003)

[BRS03] Bettstetter, C., Resta, G., Santi, P.: The node distribution of the random waypoint mobility model for wireless ad hoc networks. IEEE Transactions on Mobile Computing 2(3), 257–269 (2003)

[CMZ06] Chalopin, J., Métivier, Y., Zielonka, W.: Local computations in graphs: The case of cellular edge local computations. Fundamenta Informaticae 74(1), 85–114 (2006)

[Fer04] Ferreira, A.: Building a reference combinatorial model for MANETs. IEEE Network 18(5), 24–29 (2004); A preliminary version appeared as On models and algorithms for dynamic communication networks: The case for evolving graphs, Algotel 2002, Meze, FR

[GMMS02] Godard, E., Métivier, Y., Mosbah, M., Sellami, A.: Termination detection of distributed algorithms by graph relabelling systems. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 106–119. Springer, Heidelberg (2002)

[LMS99] Litovsky, I., Métivier, Y., Sopena, E.: Graph relabelling systems and distributed algorithms. In: World Scientific (ed.) Handbook of graph grammars and computing by graph transformation, vol. III, pp. 1–56. World Scientific, Singapore (1999)

[Lyn89] Lynch, N.: A hundred impossibility proofs for distributed computing. In: PODC 1989: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing, pp. 1–28. ACM, New York (1989)

A New Polynomial Silent Stabilizing Spanning-Tree Construction Algorithm

Alain Cournier*

MIS, Université de Picardie, 33 Rue St Leu, 80039 Amiens France
Fax : (+33) 3 22 82 54 12
alain.cournier@u-picardie.fr

Abstract. Stabilizing algorithms can automatically recover their specifications from an arbitrary configuration in finite time. They are therefore well-suited for dynamic and failure prone environments. A silent algorithm always reaches a terminal configuration in a finite time. The spanning-tree construction is a fundamental task in distributed systems which forms the basis for many other network algorithms (like Token Circulation, Routing or Propagation of Information with Feedback). In this paper we present a silent stabilizing algorithm working in n^2 steps (where n is the number of processors in the network) with a distributed daemon, without any fairness assumptions. This complexity is totally independent of the initial values present in the network. So, this improves all the previous results of the literature.

Keywords: Distributed systems, Fault-tolerance, Silent algorithms, Spanning-tree construction, Stabilization.

1 Introduction

A distributed system consists of processors (or nodes) that are pairwise connected by communication channels (or links). Using these links, processors are able to exchange information. Programming distributed systems can be realized using distributed algorithms. A distributed algorithm is a collection of local algorithms, one for each node in the network [1].

Stabilization is a nice approach to designing distributed systems that tolerate transient failures. This notion first appears in the distributed system area with the concept of self-stabilization defined by Dijkstra in 1974 [2]: a self-stabilizing algorithm, regardless of its initial state, is guaranteed to converge into the intended behavior in finite time. In 1999, another kind of stabilization, called snap-stabilization, was introduced by Bui *et al* [3]: starting from any configuration, a snap-stabilizing algorithm always behaves according to the specifications of the problem to be solved.

In distributed systems, a spanning tree is a basic tool for many complex distributed protocols. The network is formalized as a graph $G = (V, E)$ where

* Corresponding author.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 141–153, 2010.

V is the set of network nodes (vertices) and E is the set of communication links (edges) between network nodes (E is a subset of V^2). Some definitions of spanning tree can be found in the literature [4,5], let us briefly recall one of them: A graph $G' = (V', E')$ is a spanning tree of a graph $G = (V, E)$ if and only if the three following conditions hold: G' is a subgraph of G (*i.e.* $V' = V$ and $E' \subseteq E$), G' is a connected graph (*i.e.* for any pair of vertices (x, y) there exists a path between x and y) and $|E'| = |V| - 1$.

Spanning trees are often a tool for involved distributed algorithms like routing, token circulation or broadcasting messages in the networks. These algorithms either consider spanning tree as a preexistent virtual topology where computation can be done or include spanning tree construction as a subroutine of the algorithm. Messages driven protocol is the best example of the first type of algorithms since they consider that the routing table was computed before emission of the first message. The second type, can be found in [6] for token circulation or in [7] for broadcasting.

Related Work. The first self-stabilizing spanning tree construction algorithms were published in the early 1990s. One of the first papers are due to Dolev Israeli and Moran [8]. In 1991, Chen, Yu, and Huang give another distributed algorithm [9] for this problem. This algorithm was improved in [10]. More recently, we can notice some interesting algorithms for spanning trees construction used to solve either Propagation of Information with Feedback [11,12,7], or detection of cut sets [13,14]. Kosowski and Kuszner give a very nice algorithm in 2005 [15] and more recently Burman and Kutten [16] give a very nice solution in the message passing model. A very good survey can be found in [17].

Some of these algorithms are optimal in terms of rounds, but the authors do not compute the number of steps their algorithm requires during the computation.

Motivations. Roughly speaking, the complexity in terms of rounds gives interesting informations about the behavior of the low-speed processors, while the complexity in terms of steps gives an interesting information about the behavior of the whole network and so about the high-speed processors. So we need to know both complexities to get a good appreciation of the behavior of the algorithm in terms of loads of processors and bandwidth.

Contributions. In this paper we describe a stabilizing algorithm for spanning tree construction running in $\Theta(|V|)$ rounds and $\Theta(|V|^2)$ steps of computation. It improves the best previous algorithm of Kosowski and Kuszner [15] that runs in $\Theta(|V|^2 Diam(G))$ [1] steps of computation (where $Diam$ denotes the diameter of the graph). It improves the algorithm of Chen, Yu, and Huang [9] that runs in $\Omega(Max|V|^2)$(where Max denotes the maximal value of the integer variable in the initial configuration), the famous $Min + 1$ algorithm that run in $\Omega(Max|V|^2)$[2],

[1] This complexity is for a version of [15] which explicitly gives the spanning tree as an output. An examples can be found at: http://www.laria.u-picardie.fr/~cournier/MaxPlusUn.pdf

[2] See: http://www.laria.u-picardie.fr/~cournier/MinPlusUn.pdf

the algorithm of Datta, Larmore, and Vemula [18] that runs in $\Omega(|V|^3)$, and the algorithm of Arora and Gouda [19] ($\Omega(Max^2)$). Since the complexity of the algorithm remains independent of the value Max our algorithm is also more flexible since we never need a good approximation of the size of the network to initialize Max.

Outline of the paper. In Section 2 we formalize the programming model to be considered. We then give an algorithm for unbounded variables (see Section 3). In Section 4 we formally prove this first algorithm. We then conclude in the last section.

2 Preliminaries

We consider an asynchronous network $G = (V, E)$ of $|V|$ processors connected by bi-directional links according to an arbitrary topology. For an arbitrary processor p, $p.Neig$ denotes the set of neighbors of processor p. In this paper, we will deal with connected network. We assume the local shared memory model of communication. The program of every processor consists of a set of *locally shared variables* (henceforth, referred to as variables) and a finite set of actions. A processor can only write to its own variables, and read its own variables and variables owned by the neighboring processors.

Each action is of the following form: $< label >:: < guard > \longrightarrow < statement >$. The guard of an action in the program of p is a boolean expression involving the variables of p and its neighbors. The statement of an action of p updates one or more variables of p. An action can be executed only if its guard evaluates to true. We assume that the actions are atomically executed, meaning, the evaluation of a guard and the execution of the corresponding statement of an action, if executed, are done in one atomic step. The label is just a name given to the rule; this name will be useful to denote a particular action of an algorithm.

The *state* of a processor is defined by the value of its variables. The *state* of a system is the product of the states of all processors ($\in V$). We will refer to the state of a processor and system as a (*local*) *state* and (*global*) *configuration*, respectively. A processor p is said to be *enabled* in Configuration γ if there exists at least an action A such that the guard of A is true in γ. We say that processor p executes a *disable action* between configurations γ_i and γ_{i+1} if p was enabled in γ_i and not enabled in γ_{i+1}, but did not execute any action between these two configurations. (The disable action represents the following situation: At least one neighbor of p changed its state between γ_i and γ_{i+1}, and this change effectively made the guard of all actions of p false.) Similarly, an action A is said to be enabled (in γ) at p if the guard of A is true at p (in γ). A *computation step* is a transition between two configurations where the transition contains at least one action and at most one action per processor. The *distributed* daemon (respectively *central* daemon) implies that during a computation step, if one or more processors are enabled, then the daemon chooses at least one (respectively exactly one) of these enabled processors to execute an action.

In order to compute the time complexity measure, we use the definition of *round* [20]. This definition captures the execution rate of the slowest processor in any computation. Given a computation e, the *first round* of e (let us call it e') is the minimal prefix of e containing the execution of one action (an action of the protocol or the disable action) of every continuously enabled processor from the first configuration. Let e'' be the suffix of e, i.e., $e = e'e''$. Then *second round* of e is the first round of e'', and so on.

A *silent algorithm* is a distributed algorithm where any execution reaches a *terminal configuration* (a terminal configuration is a configuration where no processor is enabled).

3 The Algorithm

3.1 Algorithm Outlines

In this first approach we provide a semi uniform algorithm, in which each node has three local variables Par, L and EA. Semi-uniformity means that exactly one of the nodes, called the *root*, needs to be distinguished. We will denote this node by r. Our algorithm uses this node r as the root of the spanning tree.

For a node x we will denote by $x.Par$ (respectively $x.L$ and $x.EA$) the local variable Par (respectively L and EA) of the node x. By extension $x.Par.L$ denotes the Variable L of the parent of the node x.

For a node x, the interpretation of the variables Par and L is as follow. The variable $x.Par$ is the parent of x in the spanning tree, since r is the root of the spanning tree we introduce a special constant value $r.Par = \perp$. The variable $x.L$ is the height of the node x in the spanning tree, since r is the root of the spanning tree $r.L = 0$ is a constant value. Of course, since we describe a stabilizing algorithm, we cannot assume any particular value for these two variables. But, we can notice there is a strong constraint between them: For any node $x \neq r$ the level of the parent of x must be equal to the level of x minus 1. And more formally : $(x \neq r) \Rightarrow (x.L = x.Par.L + 1)$ (1) This simple constraint induces that for any configuration γ, we can build a covering forest $F_\gamma = (V, E_\gamma")$, where $E_\gamma" = \{(x, y) \in E | (x \neq r) \wedge x.Par = y \wedge x.L = y.L + 1\}$. In this forest let us call *abnormal root* any node x such that $x \neq r$ and the edge $(x, x.Par) \notin E_\gamma"$. Every abnormal root x can detect that it must execute an action since $(x.L \neq x.Par.L + 1)$. Generally, algorithms try to find immediately a new parent for the node x. This strategy, may lead to the execution of a great number of moves during the stabilization. To improve this strategy we chose to create a new variable dedicated to Error Administration (EA). For a node $x \neq r$, this variable $x.EA$ can take 4 values : C (Clean), EB (Error Broadcast), EF (Error Feedback) and WT (Waiting for a Tree). Since the root r does not meet any error, $r.EA = C$ is a constant. A node $x \neq r$ verifies $x.EA = C$ when it satisfies (1) and $x.Par.EA = C$. When an abnormal root x executes an action, it sets the value of its $x.EA$ variable to EB. This value will be top down propagated on the whole tree rooted by x. When a leaf of the tree receives this value it will change it to EF. This EF value will be bottom up propagated

from the leaf to the root of the subtree. When x received the value EF, x can claim that for any node y, if y is in the tree rooted by x, $y.EA \neq C$. Then x will propagate the information WT. This value will be top down propagated on the whole tree rooted by x. Then the only nodes authorized to change the value of their Par and L variables are the nodes y such that $y.EA = WT$. Of course, if a node y choses a node z as its new parent, $z.EA = C$. As a consequence, when a node x is able to choose a new parent y, we can claim that x and y are not in the same tree of the forest F. Since we want to realize a deterministic choice of a parent on a node $x \neq r$, we assume that the neighborhood of the node x ($x.Neig$) is locally ordered by an arbitrary total order $\succ_p$. Each of these orders is totally independent and does not need to induce any property of G.

Algorithm 1 page 145, is an implementation of the principles previously described.

Input: $p.Neig$: set of (locally) ordered neighbors
Constants: $p.L = 0$; $p.Par = \perp$; $p.EA = C$; if $p = r$
Variables : when $p \neq r$
$p.L$: a natural integer;
$p.Par \in p.Neig$;
$p.EA \in \{C, EB, EF, WT\}$
Macros:
$p.Potential = \{q \in p.Neig | q.EA = C\}$
$p.MinPot = \{q \in p.Potential | \forall t \in p.Potential,\ q.L \leq t.L\}$
$p.Ch = t$ where t is the minimal element of $p.MinPot$ with respect to $\succ_p$

Algorithm for $p \neq r$
Predicates:
$PED(p) \equiv p.EA = C \wedge (p.L \neq p.Par.L + 1 \vee p.Par.EA \neq C)$
$PEC(p) \equiv p.EA = EB \wedge \forall q \in p.Neig,\ ((q.Par = p \Rightarrow q.EA \neq C)$
$\wedge((q.Par = p \wedge q.EA = EB) \Rightarrow p.L \geq q.L))$
$PEE(p) \equiv p.EA = EF \wedge (p.Par.EA = EF \Rightarrow p.Par.L \geq p.L)$
$\wedge \forall q \in p.Neig,\ (q.EA \neq EB \wedge (q.Par = p \Rightarrow q.EA \in \{EF, WT\}))$
$PTC(p) \equiv p.EA = WT \wedge p.MinPot \neq \emptyset \wedge \forall q \in p.Neig,\ (q.EA \in \{C, WT\}$
$\wedge(q.Par = p \Rightarrow q.EA \neq C))$
Actions:
$EDA :: PED(p) \rightarrow p.EA \leftarrow EB;$
$ECA :: PEC(p) \rightarrow p.EA \leftarrow EF;$
$EEA :: PEE(p) \rightarrow p.EA \leftarrow WT;$
$TCA :: PTC(p) \rightarrow p.Par \leftarrow p.Ch; p.L \leftarrow p.Ch.L + 1; p.EA \leftarrow C$

Algorithm 1. A polynomial algorithm.

4 Proof Outline of Algorithm 1

The proof of the algorithm will be split into 2 parts. First we prove that any terminal configuration induces a spanning tree of the network $G = (V, E)$ (see section 4.2). We then prove in section 4.3 that any execution of Algorithm 1 reaches a terminal configuration in at most $\Theta(|V|)$ rounds and $\Theta(|V|^2)$ steps of computation.

4.1 Definitions and Notations

In the sequel, $\gamma.p.v$ denotes the local variable v of the node p in the configuration γ. We now define a particular subnetwork of G.

Definition 1. *Let $G = (V, E)$ be a network and γ be any configuration of Algorithm 1 for the network G. $G'_\gamma = (V'_\gamma, E'_\gamma)$ is the directed sub-network of G such that :*

1. $V'_\gamma = V$;
2. $\forall (x, y) \in V^2$, $(x, y) \in E'_\gamma$ *if and only if the 4 following conditions hold :*
   (a) $(x, y) \in E$; *(b)* $\gamma.x.Par = y$; *(c)* $\gamma.x.L = \gamma.y.L + 1$;
   (d) $\gamma.x.EA \neq y.EA \Rightarrow ((x.EA = C \wedge y.EA = EB) \vee (x.EA = EF \wedge y.EA \neq C))$.

We can notice that G'_γ is an antisymmetric relation. Furthermore, by transitivity, Definition 1(c) implies G'_γ does not contains cycles, and we can claim the following property.

Property 1. Let $G = (V, E)$ be a network and γ be any configuration of Algorithm 1 for the network G. G'_γ is a covering forest of G.

In the sequel we will denote by F_γ be the covering forest of G in configuration γ. Let $F_\gamma = \{T^0_\gamma, \ldots, T^t_\gamma\}$ the covering forest of G. Without lost of generality we can assume that $r \in T^0_\gamma$. Since G'_γ does not contain any cycle, for each tree T^i_γ there exists exactly one node denoted r^i_γ in T^i_γ such that $r^i_\gamma.Par \notin T^i_\gamma$, and of course $r = r^0_\gamma$.

Remark 1. Let $G = (V, E)$ be a network and γ be any configuration of Algorithm 1 for the network G. Let $F_\gamma = \{T^0_\gamma, \ldots, T^t_\gamma\}$ be the covering forest associated with γ. By construction, the four following assumptions hold:

1. $\forall i$, $\gamma.r^i_\gamma.EA = C$ implies $\forall q \in T^i_\gamma$, $\gamma.q.EA = C$;
2. $\forall i$, $\gamma.r^i_\gamma.EA = EB$ implies $\forall q \in T^i_\gamma$, $\gamma.q.EA \neq WT$;
3. $\forall i$, $\gamma.r^i_\gamma.EA = EF$ implies $\forall q \in T^i_\gamma$, $\gamma.q.EA = EF$;
4. $\forall i$, $\gamma.r^i_\gamma.EA = WT$ implies $\forall q \in T^i_\gamma$, $\gamma.q.EA \in \{EF, WT\}$.

4.2 Terminal Configurations of Algorithm 1

First we characterize some terminal configurations of the algorithm.

Property 2. Let γ be a global configuration such that for every node p, $\gamma.p.EA = C$ (C1) and $p \neq r \Rightarrow \gamma.p.L = \gamma.p.Par.L + 1$ (C2). We can then claim:

1. γ is a terminal configuration of Algorithm 1;
2. F_γ induces a spanning tree of the network.

Proof. First, let us prove that γ is a terminal configuration of Algorithm 1. Let $p \neq r$ be an arbitrary node of the network[3]. Using C1 the predicates $PEC(p)$, $PEE(p)$ and $PTC(p)$ cannot be satisfied. Furthermore using (C2) and (C1) the

[3] The case $p = r$ is obvious since there is no action for the root.

predicate $PED(p)$ cannot be satisfied. So p is disabled. Second, let us prove that γ induces a covering tree of the network. Let us represent our network by $G = (V, E)$. Let $F_\gamma = (V, E')$ be the covering forest of G (see Definition 1). Using (C1), every node p satisfies $\gamma.p.EA = C$. So for an arbitrary node p, let $\mu = p_0 \dots p_k$ be a maximal elementary path such that $p_0 = p$ and $\forall i, 0 \leq i < k, p_i.Par = p_{i+1}$. Under these assumptions $p_k = r$. Suppose the contrary ($p_k \neq r$), there exists a node q such that $\gamma.p_k.Par = q$, since μ is a maximal elementary path, there exists some j such that $q = p_j$. Using (C2) and $p_k.Par = q = p_j$ implies $p_j.L < p_k.L$ and using the same condition (C2) upon μ, $p_j.L > p_{j+1}.L > \dots > p_k.L$, a contradiction.

As a consequence, F_γ is a connected forest containing exactly $|V| - 1$ edges and by definition (see [4]) F_γ is a spanning tree of G.

Now we must prove that any terminal configuration has to satisfy Property 2. This is the aim of the following properties.

Property 3. Let γ be a global configuration such that there exists a node p which satisfies $p.EA \in \{EB, EF\}$, then γ is not a terminal configuration.

Proof. Let us suppose the contrary, let γ be a terminal configuration containing a node p such that $p.EA \in \{EB, EF\}$, since $r.EA = C$ is a constant $p \neq r$. So we must study the two following cases:
Case 1 $p.EA = EB$**:** let q be a node such that $\gamma.q.EB$ with a maximal level (*i.e.* $\forall u \in V$, $\gamma.u.EA = EB$ yields $\gamma.q.L \geq \gamma.u.L$). If q does not satisfies $\gamma.PEC(q)$ then there exists a neighbor v of q such that $\gamma.v.EA = C$ and $\gamma.v.Par = q$. Then, v satisfies $\gamma.PED(v)$ and γ is not a terminal configuration. A contradiction.
Case 2 $p.EA = EF$**:** First we notice that if there exists a node t such that $\gamma.t.EA = EB$ then using Case 1, γ is not a terminal configuration. So, in the other cases, let q be a node such that $\gamma.q.EF$ with a minimal level(*i.e.* $\forall u \in V$, $\gamma.u.EA = EF$ yields $\gamma.q.L \leq \gamma.u.L$). If q does not satisfies $\gamma.PEE(q)$ then there exists a neighbor v of q such that $\gamma.v.EA = C$ and $\gamma.v.Par = q$. Then, v satisfies $\gamma.PED(v)$ and γ is not a terminal configuration. A contradiction.

Property 4. Let G be a network and γ be a global configuration of the algorithm, such that there exists a node p which satisfies $p.EA = WT$, then γ is not a terminal configuration.

Proof. Using Property 3, we can focus on global configurations γ such that each node p satisfies $p.EA \in \{C, WT\}$. Let t be any node such that $t.EA = WT$[4]. Let $\mu = (p_0 = r, \dots, p_k = t)$ be any elementary path from r to t. Since the network is connected, such a path μ always exists. Let $1 \leq i \leq k$ be the lowest integer such that $\gamma.p_i.EA = WT$, since $r.EA = C$ and $t.EA = WT$ such a node always exists. Then $\gamma.p_{i-1}.EA = C$ so either p_i satisfies the predicate $PTC(p_i)$ or there exists $z \in Neig_{p_i}$ such that $z.EA = C$ and $z.Par = p_i$ and in this case z satisfies $PED(z)$. In both cases, γ is not a terminal configuration.

[4] $t \neq r$ since $r.EA = C$ is a constant of the algorithm.

These two properties yield that any terminal configuration only contains nodes p such that $p.EA = C$. Now we have to verify that any terminal configuration satisfies the two conditions of Property 2.

Property 5. Let γ be a terminal configuration of the algorithm on a connected network G, the two following conditions hold: 1. $\forall p$, $p.EA = C$; 2. $\forall p \neq r$, $p.L = p.Par.L + 1$.

Proof. Using Properties 3 and 4, we can focus on global configurations γ such that each node p satisfies $p.EA = C$. But in such a configuration, if there exists some nodes $p \neq r$ such that $p.L \neq p.Par.L + 1$ then p satisfies the predicate $PED(p)$ and γ is not a terminal configuration.

Theorem 1 flows from Properties 5 and 2.

Theorem 1. *Let G be a network, if Algorithm 1 reaches a terminal configuration, a spanning tree of G has been computed.*

But Theorem 1 does not yield that for every execution Algorithm 1 will reach a terminal configuration. We will prove this fact in the next section.

4.3 Convergence and Time of Stabilization

Basic properties of the forest First let us prove that for any execution starting in an arbitrary configuration γ, T^0_γ cannot loose any node during an execution.

Lemma 1. *Let γ_1 and γ_2 be two consecutive configurations of an execution E of Algorithm 1, then $T^0_{\gamma_1}$ is a subtree of $T^0_{\gamma_2}$.*

Proof. Let γ be an arbitrary configuration. Since $r.EA = C$, using Remark 1 $q \in T^0_\gamma$ yields $\gamma.q.EA = C$. Furthermore, using Definition 1 and Property 1 $\gamma.q.Par = y$ implies $y \in T^0_\gamma$ and $\gamma.q.L = \gamma.y.L + 1$. So, q is not enabled in configuration γ. So, between two consecutive configurations γ_1 and γ_2 the nodes in $T^0_{\gamma_1}$ cannot perform an action so we can claim $T^0_{\gamma_1}$ is a subtree of $T^0_{\gamma_2}$.

Let us study the evolution of a tree $T \neq T^0$ of a forest F during an execution.

Lemma 2. *Let γ_1 and γ_2 be two consecutive configurations of an execution e of Algorithm 1 and xy be an edge of F_{γ_1} then either xy is an edge of F_{γ_2} or in configuration γ_1, $x.EA = y.EA = WT$.*

Proof. First we can notice that if neither x nor y executes an action between these two configurations, (x, y) remains an edge of F_{γ_2}. So we will only focus on cases where at least one of this two nodes executes an action. Let us study the four following cases:
Case 1 : $\gamma_1.x.EA = C$. In this case, since (x, y) is an edge of F_{γ_1}, $y.EA \in \{C, EB\}$ then either $\gamma_1.y.EA = C$ and y executes the EDA action (x does not satisfies the predicate $PED(x)$) or $\gamma_1.y.EA = EB$ and x executes the EDA

action (y does not satisfies the predicate $PEC(x)$). In both cases (x, y) remains an edge of F_{γ_2};
Case 2 : $\gamma_1.x.EA = EB$**.** In this case, since (x, y) is an edge of F_{γ_1}, $\gamma_1.y.EA = EB$ and y does not satisfy the predicate $PEC(y)$. As a consequence, x executes the ECA action and (x, y) remains an edge of F_{γ_2};
Case 3 : $\gamma_1.x.EA = EF$**.** In this case, since (x, y) is an edge of F_{γ_1}, $y.EA \neq C$ then either $\gamma_1.y.EA = EF$ and y executes the EEA action (x does not satisfies the predicate $PEE(x)$) or $\gamma_1.y.EA = WT$ and x executes the EEA action (y does not satisfies the predicate $PTC(y)$) or $\gamma_1.y.EA = EB$ in this case x is not enabled and y executes the ECA action. In all cases (x, y) remains an edge of F_{γ_2};
Case 4 : $\gamma_1.x.EA = WT$**.** By construction of F_{γ_1} we can claim $x.EA = WT$ yields $y.EA = WT$.

In all cases the lemma holds.

The following corollaries follow directly from Lemma 2:

Corollary 1. *Let γ_1 and γ_2 be two consecutive configurations of an execution e of Algorithm 1 and $T^i_{\gamma_1}$ be any tree of F_{γ_1} such that $r_{T^i_{\gamma_1}}.EA \in \{C, EB, EF\}$ then there exists a tree $T^j_{\gamma_2}$ of F_{γ_2} such that $T^i_{\gamma_1}$ is a subtree of $T^j_{\gamma_2}$.*

Corollary 2. *Let γ_1 and γ_2 be two consecutive configurations of an execution e of Algorithm 1 and $T^i_{\gamma_1}$ be any tree of F_{γ_1}. If for every tree T of F_{γ_2}, $T^i_{\gamma_1}$ is not a subtree of T then $\gamma_1.r^i_{\gamma_1}.EA = WT$.*

Furthermore, if there exists a tree T of F_{γ_2} such that $T \neq T^i_{\gamma_1}$ and T is a subtree of $T^i_{\gamma_1}$, then in configuration γ_2, $r_T.EA = WT$.

Convergence and time of stabilization part 1: Number of rounds. In this section we show that Algorithm 1 stabilizes and its stabilization time is upper bounded by $4|V|$ rounds.

Theorem 2. *Algorithm 1 reaches a terminal configuration (Thus computes a spanning tree see Theorem 1) in at most $4|V|$ rounds and more generally in $\Theta(|V|)$ rounds.*

Proof Outline. A complete proof is given in appendix[5]. The main idea of the proof is that when a node p in a configuration γ may set its Parent variable to another value, $\gamma.p.EA = WT$. Let T be the tree of F_γ containing the node p. Using Remark 1 and Corollary 1 we can claim the two following assumptions:

1. $q \in T$ and $q.EA = C$ yields no node of T is able to change its variable Par;
2. $q \in T$ and $q.EA = WT$ yields any node $x \in T$ satisfies $x.EA \neq C$.

So when a node may leave its tree T, no node is able to hook on T. This is the crucial point of the proof. Then a simple induction proves that after at most $|V|$ rounds we can claim that $x.EA = C$ implies $x \in T^0$. A second simple induction shows that after at most $2|V|$ rounds we can claim that any nodes x satisfies

[5] Appendix can be found at: http://www.laria.u-picardie.fr/cournier/ApSirocco.pdf

$x.EA \neq EB$. The third one shows that after at most $3|V|$ rounds we can claim that any nodes x satisfies $x.EA \neq EF$. Then it will take at most $|V|$ more rounds to hook each node x such that $x.EA = WT$ on T^0. □

Stabilization and time of stabilization part 2: Number of steps. We now compute the number of steps required by the algorithm to reach a terminal configuration. To compute this number of steps we will associate a weight to any configuration (Definition 2). Then we will show that this weight strictly decreases for any execution using a central daemon (Property 6). We then extend this result to a distributed daemon(Corollary 3).

Definition 2. *Let γ be an arbitrary configuration of Algorithm 7 and $F_\gamma = \{T^0_\gamma, \ldots, T^k_\gamma\}$ its associated forest. For any tree T of the forest F_γ let us denote by n^C_T: the number of nodes x in T such that $x.EA = C$ (In the same way we can define n^{EB}_T, n^{EF}_T and n^{WT}_T), furthermore let $r(T)$ be the root of T. So, we can give a weight $w(T)$ to each tree T of the forest F_γ.*

- $w(T^0_\gamma) = 0$ (since $r(T^0_\gamma)$ is the root of our network);

- $w(T) = 2n^{EF}_T + n^{WT}_T$ when $r(T).EA \in \{EF, WT\}$;

- $w(T) = 4n^C_T + 3n^{EB}_T + 2n^{EF}_T + 5(|V| - (|T| + |T^0_\gamma|))$ when $T \neq T_0$ and $r(T).EA \in \{C, EB\}$.

And by extension, the weight $W(F_\gamma)$ of the forest F in configuration γ is: $W(F_\gamma) = \sum_{i=0}^{k} w(T^i_\gamma)$.

The following remark gives an upper bound for the weight of any forest.

Remark 2. Let γ be an arbitrary configuration of Algorithm 1 and F_γ its associated forest. Then $0 \leq W(F_\gamma) < 5|V|^2$. Furthermore, any terminal configuration γ satisfies $W(F_\gamma) = 0$.

Proof. First we can notice that the weight of a tree T is lower than $5|V|$ and greater than or equal to 0. Since there is at most $|V|$ trees in the forest F, $0 \leq W(F) < 5|V|^2$. Since in any terminal configuration each node is in T^0_γ (see Theorem 1), $W(F_\gamma) = 0$.

Furthermore, we can notice that the weight of a forest is totally independent from the values of variables L. In the following we will prove that this weight will strictly decrease during any execution. To do that we will show that any application of any action induces a reduction of the weight of the forest. Intuitively, a node in a dead tree[6] will execute at most two actions before it is inserted in another alive tree. On the other hand an alive tree cannot loose any node before it becomes a dead tree, furthermore any node in an alive tree can do at most 4 actions (5 with the insertion in the tree). So in worst case the algorithm stabilizes in $O(|V|^2)$ steps.

Property 6. Let γ_1 and γ_2 be two consecutive configurations of an execution using a central daemon then $W(F_{\gamma_1}) > W(F_{\gamma_2})$.

[6] A tree T is a dead tree if and only if $r(T).EA \in \{EF, WT\}$ otherwise T is alive.

Proof Outline. Let γ be a configuration and x an enabled node. One can easily verify that the execution of an action by x induces a configuration with a lower weight. A complete proof is given in appendix. □

This property implies that the algorithm stabilizes using a central demon. But the case of a distributed demon is a little bit more complicated. A way to prove the result is to show that any step of the algorithm using a distributed demon can be simulated by a central demon. To do this we just need to prove that for any group of actions that can be done in one step by a distributed demon, could be executed sequentially by a central demon also.

In order to prove this assertion, we need to define a partial order. First we need to define the following total order for the value of the variable EA: $WT < EF < EB < C$. This total order will be used to define a partial order on the nodes of the network in a configuration.

Definition 3. *Let $Q = \{x_1, \ldots, x_k\}$ be a set of enabled processor. For any elements x_i and x_j we will say that $x_i \leq_p x_j$ if and only if one of the 3 following conditions holds:*

- $i = j$;

- $x_i.EA < x_j.EA$;

- $x_i.EA = x_j.EA = WT$ and $x_i.Ch.L > x_j.Ch.L$ where $p.Ch$ is the node which can be chosen by p as its new parent during the execution of the TCA action.

Let W_γ be the set of all enabled processors in configuration γ and $Q_\gamma \subseteq W_\gamma$ be the choice of the daemon in this configuration. Then a central daemon can always executes sequentially the actions of Q_γ in the order of any linear extension of our partial order $\leq_p$.

Property 7. Let γ be any state of the algorithm, W_γ be the set of all enabled processors in configuration γ and Q_γ be any subset of W_γ. If Q_γ transforms configuration γ in configuration U in one step then a central demon can also transform configuration γ in configuration U by choosing the nodes in the order of any linear extension of $(Q_\gamma, \leq_p)$ (where $\leq_p$ is the order of Definition 3).

Proof Outline. A complete proof is given in the appendix. The idea of the proof is the following. Let γ_1 and γ_2 be two arbitrary consecutive configurations of Algorithm 1 using a distributed daemon. Let Q_{γ_1} be the set of processors that execute during this step of computation. So let p be a minimal processor of Q_{γ_1} with respect of $\leq_p$. So we can assume that γ' is the configuration deduced from γ_1 when p is the only processor that executes an action. Then we just need to verify that $W_{\gamma'}$ contains $Q_{\gamma_1} - \{p\}$. So step by step a central daemon can choose the minimal element of the set to realize the same computation. □

Corollary 3. *For any execution of Algorithm 1 the weight of state strictly decreases at each step of the execution.*

Proof. This is a direct consequence of Property 6 and Property 7.

So we can give our main result.

Theorem 3. *Algorithm 1 reaches a terminal configuration (thus compute a spanning tree) in at most* $5|V|^2$ *steps.*

Proof. This is a direct consequence of Corollary 3 and Remark 2.

5 Concluding Remarks

In this paper a stabilizing algorithm running in quadratic time to construct a spanning tree has been presented, the complexity is independent from the initial values present in the variables. Furthermore since our algorithm is silent, it can be easily implemented in the message passing model and could be useful for energy efficient protocols in sensor networks.

References

1. Tel, G.: Introduction to distributed algorithms. Cambridge University Press, Cambridge (2001)
2. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communication of the Association of the Computing Machinery 17, 643–644 (1974)
3. Bui, A., Datta, A.K., Petit, F., Villain, V.: State-optimal snap-stabilizing pif in tree networks. In: Arora, A. (ed.) WSS, pp. 78–85. IEEE Computer Society, Los Alamitos (1999)
4. Berge, C.: Graphes et hypergraphes. Dunod (1985)
5. Aho, A., Ullman, J.: Foundations of Computer Science. W.H. Freeman and Company, New York (1992)
6. Cournier, A., Devismes, S., Villain, V.: A Snap-Stabilizing DFS with a Lower Space Requirement. In: Tixeuil, S., Herman, T. (eds.) SSS 2005. LNCS, vol. 3764, pp. 33–47. Springer, Heidelberg (2005)
7. Cournier, A., Devismes, S., Villain, V.: Snap-Stabilizing PIF and Useless Computations. In: ICPADS (1), pp. 39–48. IEEE Computer Society, Los Alamitos (2006)
8. Dolev, S., Israeli, A., Moran, S.: Self-stabilization of dynamic systems assuming only read/write atomicity. Distributed Computing 7(1), 3–16 (1993)
9. Chen, N.S., Yu, H.P., Huang, S.T.: A Self-Stabilizing Algorithm for Constructing Spanning Trees. Inf. Process. Lett. 39(3), 147–151 (1991)
10. Huang, S.T., Chen, N.S.: A Self-Stabilizing Algorithm for Constructing Breadth-First Trees. Inf. Process. Lett. 41(2), 109–117 (1992)
11. Cournier, A., Datta, A.K., Petit, F., Villain, V.: Self-Stabilizing PIF Algorithm in Arbitrary Rooted Networks. In: ICDCS, pp. 91–98 (2001)
12. Cournier, A., Datta, A.K., Petit, F., Villain, V.: Snap-Stabilizing PIF Algorithm in Arbitrary Networks. In: 22th IEEE International Conference on Distributed Computing Systems (ICDCS 2002), Vienna, July 2002, pp. 199–208. IEEE Computer Society Press, Los Alamitos (2002)
13. Cournier, A., Devismes, S., Villain, V.: Snap-Stabilizing Detection of Cutsets. In: Bader, D.A., Parashar, M., Sridhar, V., Prasanna, V.K. (eds.) HiPC 2005. LNCS, vol. 3769, pp. 488–497. Springer, Heidelberg (2005)

14. Devismes, S.: A Silent Self-Stabilizing Algorithm for finding Cut-nodes and Bridges. Parallel Processing Letters 49(1-2), 183–198 (2005)
15. Kosowski, A., Kuszner, L.: A self-stabilizing algorithm for finding a spanning tree in a polynomial number of moves. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 75–82. Springer, Heidelberg (2006)
16. Burman, J., Kutten, S.: Time optimal asynchronous self-stabilizing spanning tree. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 92–107. Springer, Heidelberg (2007)
17. Gaertner, F.C.: A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms. Technical report, Swiss Federal Institute of Technology, EPFL (2003)
18. Datta, A.K., Larmore, L.L., Vemula, P.: Self-stabilizing leader election in optimal space. In: Kulkarni, S., Schiper, A. (eds.) SSS 2008. LNCS, vol. 5340, pp. 109–123. Springer, Heidelberg (2008)
19. Arora, A., Gouda, M.G.: Distributed reset. IEEE Trans. Computers 43(9), 1026–1038 (1994)
20. Dolev, S., Israeli, A., Moran, S.: Uniform dynamic self-stabilizing leader election. IEEE Trans. Parallel Distrib. Syst. 8(4), 424–440 (1997)

Spatial Node Distribution of Manhattan Path Based Random Waypoint Mobility Models with Applications

Pilu Crescenzi[1], Miriam Di Ianni[2], Andrea Marino[1], Gianluca Rossi[2], and Paola Vocca[3]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Firenze, Italy
[2] Dipartimento di Matematica, Università di Roma "Tor Vergata", Roma, Italy
[3] Dipartimento di Matematica "Ennio De Giorgi", Università del Salento, Lecce,Italy

Abstract. In this paper, we study the spatial node stationary distribution of two variations of the Random Waypoint (in short, RWP) mobility model. In particular, differently from the RWP mobility model, that connects source to destination points by straight lines, our models make use of Manhattan or (more realistically) Bezier paths. We provide analytical results for the spatial node stationary distribution for the two Manhattan based RWP mobility models and experimental evidence that the Bezier based models do not significantly differ from the Manhattan ones. This implies that Manhattan based RWP models can be considered a good approximation of the more realistic Bezier ones. As a case study, we exploit our results about one of the two Manhattan based RWP models to derive an upper bound on the transmission range of the nodes of a MANET, moving according to this model, that with high probability guarantees the connectivity of the communication graph.

1 Introduction

The *Random WayPoint* (in short, `RWP`) mobility model [12] is one of the most commonly used models for evaluating the performance of a communication protocol and/or application based on a *mobile wireless ad hoc network* (in short, `MANET`). According to this model, each node moves itself by selecting a random destination point T (within a specified movement region, which is typically a square) and a random speed value v (usually chosen uniformly within a specified interval), and by travelling from its current position S to T at constant speed v along the segment joining S to T (for a survey on mobility models for `MANET` research, see [5]).

Due to its simplicity, the `RWP` model has been widely analyzed in the literature, from both an experimental and a theoretical point of view. In particular, in the last few years several papers studied and tried to estimate the *spatial node stationary distribution* of the model [4,3,13,14]. Obtaining an exact closed formula for this distribution might turn out to be very useful in order to derive analytical results concerning, for instance, some topological properties of the communication graph of a `MANET` and the performance of specific protocols that depend on

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 154–166, 2010.

these properties. As an example, one could determine upper and lower bounds on the transmission range of the nodes of the MANET, moving according to the RWP model, that with high probability guarantees the connectivity of the communication graph (similarly to what has been done in the case of static networks in [8]), once the nodes have reached the spatial stationary distribution. More ambitiously, one could also determine upper and lower bounds on the completion time of specific information spreading protocols (such as the flooding one) as a function of the number of nodes and of the transmission range (similarly to what has been done in the case of geometric random graphs in [6,7]).

As far as we know, two main approaches have been used in the literature in order to compute a closed formula for the RWP spatial node stationary distribution. The first one is based on relatively simple geometric probability arguments [4,3,13]: however, this approach led to the necessity of computing very difficult integrals and, for this reason, allowed the authors to obtain only approximations of the exact formula (even though quite good ones). The second approach [14], instead, produces an exact formula by using a more sophisticated tool, that is, the Palm calculus which is a set of formulas that relate time averages to event averages [1] and which is not widely used or even known in applied areas.

In this paper we introduce and analyze a variation of the RWP model in which, once the source and the destination points, and the speed value have been chosen, the path followed by a node while moving from the source point to the destination one is one of the two Manhattan 2-segment paths connecting the two points (note that following a Manhattan path can be considered, in several contexts, more realistic than traveling through the rectilinear segment joining the source and the destination points). Clearly, a selection rule is needed in order to choose the path to be followed. In this paper we will consider two different selection rules: either the path is randomly chosen out of the two possible ones (in this case, the model is called *random Manhattan* RWP or, in short, rMRWP), or the path maximizing its minimum distance from the center of the movement square region is chosen (in this case, the model is called *peripheral Manhattan* RWP or, in short, pMRWP). By focusing on these two mobility models, we gain the following two advantages.

- By applying relatively simple geometric probability arguments similar to the ones used in [4,3,13], we can derive exact closed formulas for the spatial node stationary distributions of both the rMRWP and the pMRWP model. These formulas will allow us to compute an upper bound on the transmission range required for guaranteeing, with high probability, the connectivity of the communication graphs, once the network has reached the stationary distribution (this bound, in turn, can be used within frameworks for efficient broadcasting in which connectivity is one condition for guaranteeing full coverage [15]).
- The two models allow us to simulate two different mobility patterns: one that (similarly to the RWP model) induces a congestion of nodes in the center of the movement square region, and one in which nodes concentrate on a

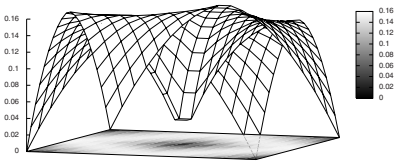


Fig. 1. The spatial node stationary distribution of the pMRWP (on the left) and of the corresponding model based on Bezier curves (on the right). The z-axis reports the $f(x, y)$ value.

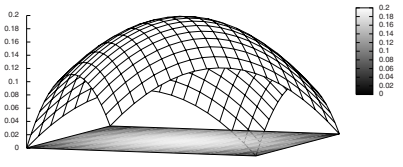
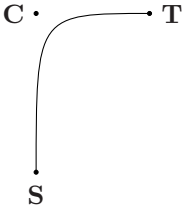


Fig. 2. The spatial node stationary distribution of the rMRWP (on the left) and of the corresponding model based on Bezier curves (on the right). The z-axis reports the $f(x, y)$ value.

peripheral ring surrounding the center of the region (hence, capturing traffic characteristics of some huge urban area, such as the Roman one).



Even though passing through a right angle at constant speed might be considered unrealistic, we claim that our results are not only interesting from a theoretical point of view but also useful from a practical point of view. To support this latter statement, indeed, we have performed extensive simulation experiments in order to compute the spatial node stationary distribution in the case in which a node follows one of the two quadratic Bezier curves [9] whose control point is the crossing point of the corresponding Manhattan path (see the figure on the left). As it can be seen from Figures 1 and 2, the distribution computed by simulations and the one deriving from our theoretical results are quite close. In other words, the rMRWP and the pMRWP mobility models can be considered as a good approximation of the corresponding mobility models based on the Bezier curves.

The paper is organized as follows. After providing, in the rest of this section, the necessary formal definitions, Sections 2 and 3 are devoted to the analytical study of the spatial node stationary distribution of the rMRWP and the pMRWP models. In particular, in Sections 2 we describe our general approach to the problem, while in Section 3 we derive the explicit closed formulas for the spatial node stationary distribution of the two models. Successively, in Section 4 we exploit our analytical results on the pMRWP model to derive an upper bound on the transmission range to be assigned to the nodes of a MANET, moving according

to the pMRWP model, in order to guarantee with high probability the connectivity of its communication graph. Finally, we conclude in Section 5.

1.1 Formal Definitions

Let $S = (x_S, y_S)$ and $T = (x_T, y_T)$ be two points in the square $\mathcal{Q} = [0, 2a] \times [0, 2a]$ of center $Z = (a, a)$. The Manhattan paths from S to T are $m_{hv}(S, T)$ and $m_{vh}(S, T)$, where $m_{hv}(S, T)$ is the horizontal path from S to $H = (x_T, y_S)$ followed by the vertical path from H to T, and $m_{vh}(S, T)$ is the vertical path from S to $V = (x_S, y_T)$ followed by the horizontal path from V to T.

The mobility models studied in this paper are all derived by the Random Waypoint mobility model. Each node is initially positioned at a point S, randomly chosen within $\mathcal{Q}$. Successively, the node chooses a random destination point $T \in \mathcal{Q}$, and a random speed value $v \in [v_{\min}, v_{\max}]$ with $v_{\min} > 0$. Then,

- in the rMRWP mobility model the node travels at constant speed v along a path randomly chosen between $m_{hv}(S, T)$ and $m_{vh}(S, T)$;
- in the pMRWP mobility model travels at constant speed v along the Manhattan path from S to T which maximizes the minimum distance from Z.

Once the destination point T is reached, the node immediately starts the traveling process again.[1]

2 The General Approach

In this section, we briefly describe the approach that will be followed while deriving the explicit formula for the spatial node stationary distribution of the Manhattan based mobility models. Observe that, since nodes move independently, we can limit ourselves to analyze the movement of a single node.

For any p and q with $0 \leq p, q \leq 2a$, let $\mathcal{R}$ be the rectangle $[0, q] \times [0, p]$, and let $F = (q, p)$ be the top right vertex of $\mathcal{R}$. Moreover, let X be the random variable describing the location of the node, and let T and $T_{q,p}$ be the two random variables describing, respectively, the time spent by the node while moving between the source and the destination point and the time spent within $\mathcal{R}$ while moving between the two points. As proved in [3],

$$P(X \in \mathcal{R}) = \frac{E[T_{q,p}]}{E[T]}.$$

Since the speed value is chosen randomly in the interval $[v_{\min}, v_{\max}]$ with $v_{\min} > 0$, we have that $\frac{E[T_{q,p}]}{E[T]} = \frac{E[L_{q,p}]}{E[L]}$, where $E[L_{q,p}]$ denotes the expected length of

[1] In the RWP mobility model it is also assumed that, once a node reaches its destination, it stays there for a pause time t_p, randomly chosen within a specified interval. In this paper, we assume that $t_p = 0$: the case in which $t_p > 0$ can be dealt similarly to what has been done in [4].

the intersection between the node path and $\mathcal{R}$ during one movement period and $E[L]$ denotes the expected length of the node path during one movement period. In other words, the problem of computing the cumulative distribution function[2] (in short, cdf) $P(X \in \mathcal{R})$ has been reduced to the problem of computing the values $E[L_{q,p}]$ and $E[L]$. This will be our task in the next section.

Let σ denote the random variable describing the location of a node at the beginning of its movement period and τ denote the random variable describing the location of the same node at the end of the same movement period. The computation of $E[L_{q,p}]$ will be performed by distinguishing the three cases in which *(i)* both σ and τ are contained in $\mathcal{R}$ or *(ii)* σ is contained in $\mathcal{R}$ while τ is outside of $\mathcal{R}$ (or vice versa) or *(iii)* both σ and τ are outside of $\mathcal{R}$. Hence,

$$\begin{aligned} E[L_{q,p}] &= \int_{S\in\mathcal{Q}} \int_{T\in\mathcal{Q}} f(S)f(T)l(S,T,\mathcal{R})dSdT \\ &= \frac{1}{16a^4}\left(\int_{S\in\mathcal{R}} \int_{T\in\mathcal{R}} l(S,T,\mathcal{R})dSdT + \int_{S\in\mathcal{R}} \int_{T\notin\mathcal{R}} l(S,T,\mathcal{R})dSdT \right. \\ &\left. + \int_{S\notin\mathcal{R}} \int_{T\in\mathcal{R}} l(S,T,\mathcal{R})dSdT + \int_{S\notin\mathcal{R}} \int_{T\notin\mathcal{R}} l(S,T,\mathcal{R})dSdT\right) \end{aligned}$$

where f denotes the probability density function[3] (in short, pdf) of a point's location and $l(S,T,\mathcal{R})$ denotes the expected length of the intersection between the chosen path from S to T and $\mathcal{R}$. Since $E[L] = \frac{4}{3}a$ (see [10]), we get

$$P(X \in \mathcal{R}) = \frac{3}{64a^5}\left(\Gamma_1 + \Gamma_2 + \Gamma_3 + \Gamma_4\right), \tag{1}$$

where Γ_i is the i-th integral of the previous equation.

3 Manhattan Path Based Random Waypoint Mobility Models

In this section, we will make use of the following notation: given two points $\alpha = (x_\alpha, y_\alpha)$ and $\beta = (x_\beta, y_\beta)$, $\Delta_x^{\alpha,\beta} = x_\alpha - x_\beta$ and $\Delta_y^{\alpha,\beta} = y_\alpha - y_\beta$. Let $\pi(S,T,\mathcal{R})$ be a predicate that will be used to describe the (relative) positions of S and T with respect to $\mathcal{R}$. We denote by $\lambda(\pi(S,T,\mathcal{R}))$ the expected length of the intersection between the chosen Manhattan path and $\mathcal{R}$, that is, $l(S,T,\mathcal{R})$, whenever $\pi(S,T,\mathcal{R})$ is true.

[2] The cumulative distribution function completely describes the probability distribution of a random variable: in our case, for every two real numbers q and p with $0 \le q,p \le 2a$, the cumulative distribution function of X is given by $P(X \in \mathcal{R}) = P(x_X \le q \wedge y_X \le p)$.

[3] The probability density function of a random variable is a function which describes the density of probability at each point in the sample space: it is well-known that this function is the derivative of the cdf.

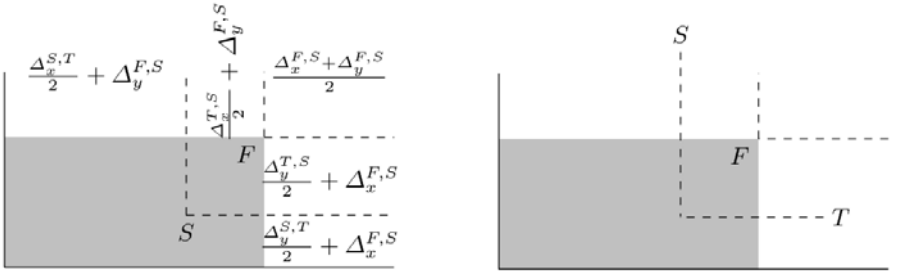


Fig. 3. The second and the third cases of the `rMRWP` mobility model

3.1 The Spatial Node Distribution of the Random Manhattan RWP Model

Computing Γ_1: $S, T \in \mathcal{R}$. This case corresponds to computing the sum of the Manhattan distances between any pair of points within a rectangle: by using the results of [10], we have that $\Gamma_1 = p^2q^2\frac{p+q}{3}$.

Computing Γ_2 (or Γ_3): $S \in \mathcal{R}$ and $T \notin \mathcal{R}$ (or vice versa). Due to symmetry reasons, we can limit ourselves to the case in which $S \in \mathcal{R}$ and $T \notin \mathcal{R}$. According to the left part of Figure 3, it is easy to compute $\lambda(S \in \mathcal{R} \wedge T \notin \mathcal{R})$: for example, if T is above S and on its left, then $\lambda(S \in \mathcal{R} \wedge T \notin \mathcal{R} \wedge x_T \leq x_S \wedge y_T \geq p) = \frac{(\Delta_x^{S,T}+\Delta_y^{F,S})+\Delta_y^{F,S}}{2} = \frac{\Delta_x^{S,T}}{2} + \Delta_y^{F,S}$. We can deal with the other four cases similarly (see the figure). Hence,

$$
\begin{aligned}
\Gamma_2 = &\int_0^q \int_0^p \left[\int_0^{x_s} \int_p^{2a} \left(\frac{x_s - x_t}{2} + p - y_s \right) dy_t dx_t dy_s dx_s \right. \\
&+ \int_{x_s}^q \int_p^{2a} \left(\frac{x_t - x_s}{2} + p - y_s \right) dy_t dx_t dy_s dx_s \\
&+ \int_q^{2a} \int_p^{2a} \frac{q - x_s + p - y_s}{2} dy_t dx_t dy_s dx_s \\
&+ \int_q^{2a} \int_{y_s}^p \left(\frac{y_t - y_s}{2} + q - x_s \right) dy_t dx_t dy_s dx_s \\
&\left. + \int_q^{2a} \int_0^{y_s} \left(\frac{y_s - y_t}{2} + q - x_s \right) dy_t dx_t dy_s dx_s \right].
\end{aligned}
$$

After having evaluated the above integral, we obtain

$$
\Gamma_2 = \Gamma_3 = a^2p^2q - \frac{1}{6}ap^3q + a^2pq^2 + ap^2q^2 - \frac{5p^3q^2}{12} - \frac{1}{6}apq^3 - \frac{5p^2q^3}{12}.
$$

Computing Γ_4: $S, T \notin \mathcal{R}$. Observe that only two situations giving raise to a non empty intersection with $\mathcal{R}$ may occur: either S is in the region above $\mathcal{R}$ and T is in the region at its right or vice versa. Since the two cases are symmetric, we

can limit ourselves to analyze only the first one (see the right part of Figure 3). Clearly, for any such pair of points S and T,

$$\lambda(x_S \leq q \wedge y_S \geq p \wedge x_T \geq q \wedge y_T \leq p) = \frac{(\Delta_x^{F,S} + \Delta_y^{F,T}) + 0}{2} = \frac{\Delta_x^{F,S} + \Delta_y^{F,T}}{2}.$$

Hence,

$$\Gamma_4 = \int_0^q \int_p^{2a} \int_q^{2a} \int_0^p \frac{q - x_s + p - y_t}{2} dy_t dx_t dy_s dx_s.$$

After having evaluated the above integral and doubled the result, we obtain

$$\Gamma_4 = \frac{1}{2}(2a - p)p(2a - q)q(p + q).$$

The spatial node distribution. After performing a normalization (that is, for the sake of simplicity, after setting $a = 1$), by applying Equation (1) we get

$$P(X \in \mathcal{R}) = \frac{1}{16} pq \left(3p - p^2 + 3q - q^2\right).$$

The pdf can be finally obtained by computing the derivative of the cdf (in doing so, for the sake of readability, we replace q and p with x and y, respectively). We have that

$$f(x, y) = \frac{3}{16} \left(2x - x^2 + 2y - y^2\right).$$

The left part of Figure 2 shows the behavior of $f(x, y)$. Observe how the central part of the domain square is visited more often due to the border effect of the model [2] (similarly to the standard `RWP` model).

3.2 The Spatial Node Distribution of the Peripheral MRWP Model

Due to the symmetry of the mobility model we consider, in this section we can limit ourselves to compute $E(L_{q,p})$ for $a > q > p$. Indeed, once we have computed the cdf and, hence, the pdf $f^{a>q>p}(x, y)$ relative to this case, we derive the global pdf in the following way:

$$f(x, y) = \begin{cases} f^{a>q>p}(y, x) & \text{if } 0 \leq x \leq a \text{ and } x \leq y \leq a, \\ f^{a>q>p}(x, y) & \text{if } 0 \leq x \leq a \text{ and } 0 \leq y \leq x, \\ f^{a>q>p}(2a - x, y) & \text{if } a \leq x \leq 2a \text{ and } 0 \leq y \leq 2a - x, \\ f^{a>q>p}(y, 2a - x) & \text{if } a \leq x \leq 2a \text{ and } 2a - x \leq y \leq a, \\ f(x, 2a - y) & \text{if } 0 \leq x \leq 2a \text{ and } a \leq y \leq 2a. \end{cases} \quad (2)$$

Computing Γ_1: $S, T \in \mathcal{R}$. This case is identical to the corresponding case of the analysis of the `rMRWP` model: hence, $\Gamma_1 = p^2 q^2 \frac{p+q}{3}$.

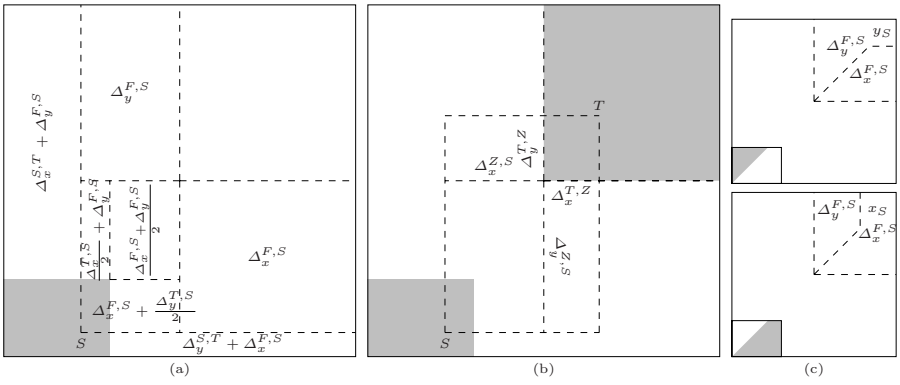


Fig. 4. The second case of the pMRWP model: S in $\mathcal{R}$ and T outside of $\mathcal{R}$

Computing Γ_2 (or Γ_3): $S \in \mathcal{R}$ and $T \notin \mathcal{R}$ (or vice versa). Once again, , we can limit ourselves to the case in which $S \in \mathcal{R}$ and $T \notin \mathcal{R}$. Let us partition the domain square into $\mathcal{Q}_1 = [0, a] \times [0, a]$, $\mathcal{Q}_2 = [a, 2a] \times [0, a]$, $\mathcal{Q}_3 = [a, 2a] \times [a, 2a]$ and $\mathcal{Q}_4 = [0, a] \times [a, 2a]$. Since $a > q > p$, both F and S certainly belong to $\mathcal{Q}_1$. If $T \in \mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \mathcal{Q}_4$, then the Manhattan path between S and T which maximizes the minimum distance from Z can be easily computed (see Figure 4(a)): for example, if T is above S and on its left, then the path which maximizes the minimum distance from Z is $m_{hv}(S, T)$, and $\lambda(S \in \mathcal{R} \wedge x_T \leq x_S \wedge y_T \geq p) = \Delta_x^{S,T} + \Delta_y^{F,S}$. Similarly we can deal with the other six cases:

$$\begin{cases} \lambda(S \in \mathcal{R} \wedge x_T \leq x_S \wedge y_T \geq p) = \Delta_x^{S,T} + \Delta_y^{F,S}, \\ \lambda(S \in \mathcal{R} \wedge x_S \leq x_T \leq a \wedge a \leq y_T \leq 2a) = \Delta_y^{F,S}, \\ \lambda(S \in \mathcal{R} \wedge x_S \leq x_T \leq q \wedge p \leq y_T \leq a) = \frac{\Delta_x^{T,S}}{2} + \Delta_y^{F,S}, \\ \lambda(S \in \mathcal{R} \wedge q \leq x_T \leq a \wedge p \leq y_T \leq a) = \frac{\Delta_x^{F,S} + \Delta_y^{F,S}}{2}, \\ \lambda(S \in \mathcal{R} \wedge a \leq x_T \wedge y_S \leq y_T \leq a) = \Delta_x^{F,S}, \\ \lambda(S \in \mathcal{R} \wedge q \leq x_T \leq a \wedge y_S \leq y_T \leq p) = \Delta_x^{F,S} + \frac{\Delta_y^{T,S}}{2}, \\ \lambda(S \in \mathcal{R} \wedge q \leq x_T \wedge y_T \leq y_S \leq p) = \Delta_y^{S,T} + \Delta_x^{F,S}. \end{cases}$$

It then remains to analyze the case in which $T \in \mathcal{Q}_3$. To this aim, we have to solve the inequality

$$\pi_1(S, T) = \min\{\Delta_x^{Z,S}, \Delta_y^{T,Z}\} < \min\{\Delta_y^{Z,S}, \Delta_x^{T,Z}\}$$

subject to the following constraints (see Figure 4(b)):

$$\pi_2(S, T, \mathcal{R}) = (p \leq q \leq a) \wedge (x_s \leq q) \wedge (y_s \leq p) \wedge (a \leq x_T) \wedge (a \leq y_T).$$

Indeed, when T satisfies the above system of linear inequalities, then the Manhattan path between S and T which maximizes the minimum distance from Z is $m_{hv}(S, T)$. By solving such system, we get (see Figure 4(c))

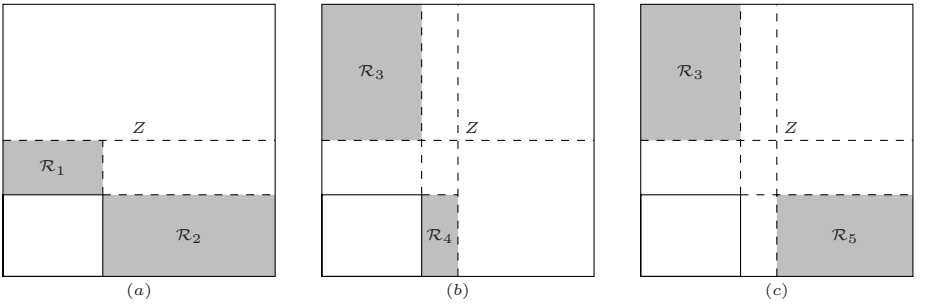


Fig. 5. The third case of the pMRWP model: both S and T outside of $\mathcal{R}$

$$\pi_1(S,T) \wedge \pi_2(S,T,\mathcal{R}) =$$
$$\begin{cases} [(0 < x_S < p) \wedge (0 < y_S < x_S) \wedge (a < x_T \leq 2a - x_S) \wedge (a < y_T < x_T)] \bigvee \\ [(0 < x_S < p) \wedge (0 < y_S < x_S) \wedge (2a - x_S < x_T < 2a) \wedge (a < y_T < 2a)] \bigvee \\ [(0 < x_S < p) \wedge (x_S < y_S < p) \wedge (a < x_T \leq 2a - y_S) \wedge (a < y_T < x_T)] \bigvee \\ [(0 < x_S < p) \wedge (x_S < y_S < p) \wedge (2a - y_S < x_T < 2a) \wedge (a < y_T < 2a - y_S)] \bigvee \\ [(p < x_S < q) \wedge (0 < y_S < p) \wedge (a < x_T \leq 2a - x_S) \wedge (a < y_T < x_T)] \bigvee \\ [(p < x_S < q) \wedge (0 < y_S < p) \wedge (2a - x_S < x_T < 2a) \wedge (a < y_T < 2a)] . \end{cases}$$

Hence, $\lambda(\pi_1(S,T) \wedge \pi_2(S,T,\mathcal{R})) = \Delta_x^{F,S}$, while $\lambda(\neg\pi_1(S,T) \wedge \pi_2(S,T,\mathcal{R})) = \Delta_y^{F,S}$.

In conclusion, Γ_2 (and, hence, Γ_3) is computed by integrating these values of $\lambda(\cdot)$ in the corresponding regions. In particular, we obtain that

$$\begin{aligned} \Gamma_2 = \Gamma_3 &= \frac{1}{12}pq\left(9a^2(p+q) - 6pq(p+q) + 2a\left(p^2+3pq+q^2\right)\right) \\ &\quad + \frac{1}{120}p\left(9p^4 - 20p^3q + 5\left(6a^2q^2+q^4\right)\right) + \frac{1}{60}p^2\left(2p^3 + 15a^2q - 5q^3\right) \\ &= \frac{1}{120}p\left(13p^4 - 20p^3q + 20p^2(a-3q)q + 10pq\left(12a^2 + 6aq - 7q^2\right)\right. \\ &\quad \left. + 5q^2\left(24a^2 + 4aq + q^2\right)\right) . \end{aligned}$$

Computing Γ_4: $S, T \notin \mathcal{R}$. Similarly to the analysis of the `rMRWP` model, only two situations may occur: either S is in the region above $\mathcal{R}$ and T is in the region at its right or vice versa. Since the two cases are symmetric, we can limit ourselves to analyze only the first one. To this aim let us consider Figure 5. If S is contained in $\mathcal{R}_1$ and T is contained in $\mathcal{R}_2$ (see Figure 5(a)), then the Manhattan path between S and T which maximizes the minimum distance from Z is $m_{vh}(S,T)$ and $\lambda(S \in \mathcal{R}_1 \wedge T \in \mathcal{R}_2) = \Delta_x^{F,S} + \Delta_y^{F,T}$. The same holds in the case in which S is contained in $\mathcal{R}_3$ and T is contained in $\mathcal{R}_4$ (see Figure 5(b)). It remains to analyze the case in which S is contained in $\mathcal{R}_3$ and T is contained in $\mathcal{R}_5$ (see Figure 5(c)). Similarly to what we have done while computing Γ_2, we have to solve the following system:

$$\begin{cases} \pi_3(S,T) = \min\{\Delta_x^{Z,S}, \Delta_y^{Z,T}\} > \min\{\Delta_y^{S,Z}, \Delta_x^{T,Z}\}, \\ \pi_4(S,T,\mathcal{R}) = (x_S \leq q) \wedge (a \leq y_S) \wedge (a \leq x_T) \wedge (y_T \leq p). \end{cases}$$

Indeed, when T satisfies the above system of linear inequalities, the Manhattan path between S and T which maximizes the minimum distance from Z is $m_{vh}(S,T)$ and $\lambda(\pi_3(S,T) \wedge \pi_4(S,T,\mathcal{R})) = \Delta_x^{F,S} + \Delta_y^{F,T}$, while $\lambda(\neg\pi_3(S,T) \wedge \pi_4(S,T,\mathcal{R})) = 0$.

In conclusion, Γ_4 is computed by integrating these values of $\lambda(\cdot)$ in the corresponding regions. By doubling the result, we obtain that

$$\begin{aligned} \Gamma_4 = \frac{1}{30} p \big(& p^4 - 10p^3 q + 30p^2 q(-2a+q) + 20pq\left(6a^2 - 6aq + q^2\right) \\ & -5q^2\left(-24a^2 + 12aq + q^2\right)\big). \end{aligned}$$

The spatial node distribution. After performing the normalization, by summing up the values Γ_1, Γ_2, Γ_3 and Γ_4 previously computed, by applying Equation (1), we obtain the cdf in the case in which $0 \leq q \leq a$ and $0 \leq p \leq q$. In particular, we have that in this case

$$\begin{aligned} P(X \in \mathcal{R}) = \frac{1}{256} p \big(& -q^4 + 3p^4 - 2q^3(10+p) + 4q^2\left(18 - 9p + p^2\right) \\ & -4qp\left(-18 + 5p + 2p^2\right)\big). \end{aligned}$$

The pdf $f^{a>q>p}(x,y)$ can be obtained by computing the derivative of the cdf (once again, for the sake of readability, we replace q and p with x and y, respectively). We have that

$$\begin{aligned} f^{a>q>p}(x,y) = \frac{1}{64} \big(& -x^3 - 3x^2(5+y) + y\left(36 - 15y - 8y^2\right) \\ & +6x\left(6 - 6y + y^2\right)\big) \end{aligned}$$

Finally, by using Equation (2), we can derive the global pdf $f(x,y)$ in the entire square region $\mathcal{Q}$.

4 Connectivity in the Peripheral Model

In this section, we exploit our analytical results on the pMRWP model to derive the transmission range to be assigned to the nodes of a MANET, moving according to this model, in order to guarantee its connectivity with high probability (note that similar results can be obtained in the case of the rMRWP model). To this aim we will follow the same approach which has been used in the case of (static) geometric random graphs (see, for example, [11]).

Let n be the number of nodes, and let $r(n) = \gamma\left(\frac{\ln n}{n}\right)^{1/3}$ be the transmission range of the nodes of the MANET (the value of γ will be specified later). By tessellating the square $\mathcal{Q}$ into k^2 square cells of size $z = 2/k$, with $k = \sqrt{5}/r(n)$, it is not difficult to prove that, in order to guarantee the connectivity of the

communication graph,[4] it suffices to choose γ so that, with high probability, every cell is not empty. For any cell C of $\mathcal{Q}$, let $X_i(C)$ be the random variable whose value is 1 if node i is in C and 0 otherwise. Moreover, let $X(C) = X_1(C) + \ldots + X_n(C)$ be the random variable describing the total number of nodes in C. By looking at the analytical expression of the pdf relative to the pMRWP model derived in the previous section (and as it clearly appears by observing the left part of Figure 11), it follows that only the cell in the center of $\mathcal{Q}$ (that is, the cell containing Z) and the cells at its corners have to be analyzed. Let C^c and C^b be, respectively, the cell in the center of $\mathcal{Q}$ and one of the cells at its corners. Then[5],

$$P[X_i(C^c) = 1] = 8 \int_{1-\frac{z}{2}}^{1} \int_{1-\frac{z}{2}}^{x} f_X(x,y)dxdy = \frac{41}{64}z^3 + z^4\frac{5z-110}{512}$$

and

$$P[X_i(C^b) = 1] = 2 \int_{0}^{z} \int_{0}^{x} f_X(x,y)dxdy = \frac{1}{32}\left(36z^3 - z^5 - 19z^4\right).$$

It is easy to verify that, since $z \leq 1$, $P[X_i(C^c) = 1] \leq P[X_i(C^b) = 1]$. Moreover,

$$\begin{aligned} P[X_i(C^c) = 1] &= \frac{41}{64}z^3 + z^4\frac{5z-110}{512} = \frac{41}{64}z^3 + z^3\left(z\frac{5z-110}{512}\right) \\ &\geq \frac{41}{64}z^3 - z^3\frac{105}{512} = \frac{223}{128}z^3. \end{aligned}$$

If we set $c = \frac{223}{128}$ and $\alpha = ncz^3$, then, for all cells C in $\mathcal{Q}$, $\mu(C) \geq \alpha$. Let D be the event "The communication graph is not connected": then, by observing that $\alpha > 1$ and by applying the Chernoff bound, we have that

$$\begin{aligned} P[D] &\leq \sum_{\text{all } C} P[X(C) < 1] < \sum_{\text{all } C} \mu(C)e^{1-\mu(C)} \leq k^2\alpha e^{1-\alpha} \\ &< \frac{20}{\gamma^2}\left(\frac{n}{\ln n}\right)^{2/3}\left(nc\frac{\gamma^3}{5^{3/2}}\frac{\ln n}{n}\right)e^{1-cn\frac{\gamma^3}{5^{3/2}}\frac{\ln n}{n}} \\ &= \frac{20}{\gamma^2}\left(\frac{n}{\ln n}\right)^{2/3}\left(c\frac{\gamma^3}{5^{3/2}}\ln n\right)\frac{e}{n^{\frac{c\gamma^3}{5^{3/2}}}} \\ &= \frac{20}{\gamma^2}\frac{n^{2/3}}{(\ln n)^{1/3}}\left(c\frac{\gamma^3}{5^{3/2}}\right)\frac{e}{n^{\frac{c\gamma^3}{5^{3/2}}}}. \end{aligned}$$

By setting $\gamma > 1.62$ we have that $\frac{c\gamma^3}{5^{3/2}} > \frac{2}{3}$, which implies that the communication graph is connected with high probability.

[4] The communication graph induced by the n nodes and the range $r(n)$ is the graph of n nodes such that an edge connecting a pair of nodes exists if and only if their distance is at most r.

[5] Observe that the worst case happens when the cell in C^c is centered in Z.

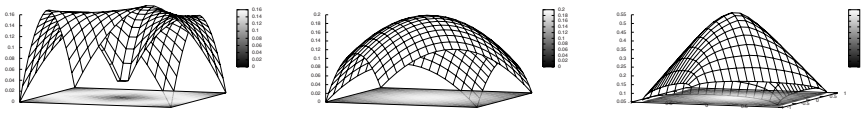


Fig. 6. The spatial node stationary distribution of the pMRWP model (left), of the rMRWPmodel (center), and of the classic RWP model (right)

5 Conclusion and Further Research

In this paper, we have analyzed the spatial node stationary distribution of two Manhattan path based variations of the RWP mobility model: in Figure 6 we compare the spatial node stationary distribution of these two models with the one of the standard RWP model. We have then applied these analytical results to the computation of an upper bound on the transmission range guaranteeing the connectivity of the communication graph of a MANET, whose nodes move according to the peripheral model.

From a mobility model point of view, it would be interesting to see whether the geometric probability arguments that we have used for the analysis of the rMRWP and the pMRWP models can be applied to the RWP model. Even though quite complicated integrals might arise, we believe it would be worth deeply analyzing this approach. From an algorithmic point of view, instead, an interesting open question concerns the possibility of obtaining analytical results about the completion time of specific information spreading protocols based on MANETs whose nodes move according to one of the mobility models analyzed in this paper. Finally, it would be interesting to compute a lower bound on the transmission range guaranteeing the connectivity of the communication graph and to see whether it matches our upper bound.

References

1. Baccelli, F., Brémaud, P.: Palm Probabilities and Stationary Queues. Springer, Heidelberg (1987)
2. Bettstetter, C.: Mobility Modeling in Wireless Networks: Categorization, Smooth Movement, and Border Effects. ACM Mobile Computing and Communications Review 5, 55–67 (2001)
3. Bettstetter, C., Resta, G., Santi, P.: The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks. IEEE Transactions on Mobile Computing 2, 257–269 (2003)
4. Bettstetter, C., Wagner, C.: The Spatial Node Distribution of the Random Waypoint Mobility Model. In: WMAN 2002, March 25 - 26, pp. 41–58 (2002)
5. Camp, T., Boleng, J., Davies, V.: A Survey of Mobility Models for Ad Hoc Network Research. Wireless Communication and Mobile Computing 2, 483–502 (2002)
6. Clementi, A., Monti, A., Pasquale, F., Silvestri, R.: Information Spreading in Stationary Markovian Evolving Graphs. In: IPDPS 2009 (to appear, 2009)

7. Clementi, A., Pasquale, F., Silvestri, R.: MANETS: High mobility can make up for low transmission power. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 387–398. Springer, Heidelberg (2009)
8. Ellis, R.B., Jia, X., Yan, C.: On random points in the unit disk. Random Structures and Algorithms 29, 14–25 (2005)
9. Farin, G.: Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide. Academic Press, London (1996)
10. Gaboune, B., Laporte, G., Soumis, F.: Expected Distances between Two Uniformly Distributed Random Points in Rectangles and Rectangular Parallelepipeds. Journal of the Operational Research Society 44, 513–519 (1993)
11. Gupta, P., Kumar, P.R.: Critical power for asymptotic connectivity in wireless networks. In: McEneaney, W.M., Yin, G.G., Zhang, Q. (eds.) Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of W. H. Fleming 1999, pp. 547–566. Birkhäuser, Basel (1999)
12. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: Imielinski, T., Korth, H. (eds.) Mobile Computing. Kluwer Academic Publishers, Dordrecht (1996)
13. Hyytia, E., Lassila, P., Virtamo, J.: Spatial Node Distribution of the Random Waypoint Mobility Model with Applications. IEEE Transactions on Mobile Computing 5, 680–694 (2006)
14. Le Boudec, J.: Understanding the simulation of mobility models with Palm calculus. Performance Evaluation 64, 126–147 (2007)
15. Wu, J., Dai, F.: Efficient broadcasting with guaranteed coverage in mobile ad hoc networks. IEEE Transactions on Mobile Computing 4, 259–270 (2005)

More Efficient Periodic Traversal in Anonymous Undirected Graphs

Jurek Czyzowicz[1], Stefan Dobrev[2], Leszek Gąsieniec[3,⋆], David Ilcinkas[4,⋆⋆], Jesper Jansson[5,⋆⋆⋆], Ralf Klasing[4], Ioannis Lignos[6], Russell Martin[3], Kunihiko Sadakane[7], and Wing-Kin Sung[8]

[1] Département d'Informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada
jurek@uqo.ca

[2] Institute of Mathematics, Slovak Academy of Sciences, Dubravska 9, P.O.Box 56, 840 00, Bratislava, Slovak Republic
stefan@ifi.savba.sk

[3] Department of Computer Science, University of Liverpool, Ashton Street, Liverpool, L69 3BX, U.K.
{L.A.Gasieniec,Russell.Martin}@liverpool.ac.uk

[4] LaBRI, CNRS and Université de Bordeaux, 351 cours de la Liberation, 33405 Talence, France
{david.ilcinkas,ralf.klasing}@labri.fr

[5] Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan
Jesper.Jansson@ocha.ac.jp

[6] Department of Computer Science, Durham University, South Road, Durham, DH1 3LE, UK
i.m.lignos@durham.ac.uk

[7] Principles of Informatics Research Division, National Institute of Informatics 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
sada@nii.ac.jp

[8] Department of Computer Science, National University of Singapore, 3 Science Drive 2, 117543 Singapore
ksung@comp.nus.edu.sg

Abstract. We consider the problem of *periodic graph exploration* in which a mobile entity with (at most) constant memory, *an agent*, has to visit all n nodes of an arbitrary undirected graph G in a periodic manner. Graphs are supposed to be anonymous, that is, nodes are unlabeled. However, while visiting a node, the robot has to distinguish between edges incident to it. For each node v the endpoints of the edges incident to v are uniquely identified by different integer labels called *port numbers*. We are interested in the minimisation of the length of the exploration period.

⋆ L. Gąsieniec partially funded by the Royal Society International Joint Project, IJP - 2007/R1. R. Martin partially funded by the Nuffield Foundation grant NAL/32566, "The structure and efficient utilization of the Internet and other distributed systems".

⋆⋆ Supported in part by the ANR projects ALADDIN and ALPAGE, the INRIA project CEPAGE, and the European projects GRAAL and DYNAMO.

⋆⋆⋆ Funded by the Special Coordination Funds for Promoting Science and Technology.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 167–181, 2010.

This problem is unsolvable if the local port numbers are set arbitrarily, see [1]. However, surprisingly small periods can be achieved when assigning carefully the local port numbers. Dobrev et al. [2] described an algorithm for assigning port numbers, and an oblivious agent (i.e., an agent with no persistent memory) using it, such that the agent explores all graphs of size n within period $10n$. Providing the agent with a constant number of memory bits, the optimal length of the period was proved in [3] to be no more than $3.75n$ (using a different assignment of the port numbers). In this paper, we improve both these bounds. More precisely, we show a period of length at most $4\frac{1}{3}n$ for oblivious agents, and a period of length at most $3.5n$ for agents with constant memory. Finally, we give the first non-trivial lower bound, $2.8n$, on the period length for the oblivious case.

1 Introduction

Efficient search in unknown or unmapped environments is one of the fundamental problems in algorithmics. Its applications range from robot navigation in hazardous environments to rigorous exploration (and, e.g., indexing) of data available on the Internet. Due to a strong need to design simple and cost effective agents as well as to design exploration algorithms that are suitable for rigorous mathematical analysis, it is of practical importance to limit the local memory of agents.

We consider the task of graph exploration by a mobile entity equipped with small (constant number of bits) memory. The mobile entity may be, e.g., an autonomous piece of software navigating through an underlying graph of connections of a computer network. The mobile entity is expected to visit all nodes in the graph in a periodic manner. For the sake of simplicity, we call the mobile entity an *agent* and model it as a finite state automaton. The task of periodic traversal of all nodes of a network is particularly useful in network maintenance, where the status of every node has to be checked regularly.

We consider here undirected graphs that are anonymous, i.e., the nodes in the graph are neither labelled nor colored. To enable the agent to distinguish the different edges incident to a node, edges at a node v are assigned *port numbers* in $\{1, \ldots, d_v\}$ in a one-to-one manner, where d_v is the degree of node v.

We model agents as *Mealy automata*. The Mealy automaton has a finite number of states and a transition function f governing the actions of the agent. If the automaton enters a node v of degree d_v through port i in state s, it switches to state s' and exits the node through port i', where $(s', i') = f(s, i, d_v)$. The memory size of an agent is related to its number of states; more precisely it equals the number of bits needed to encode these states. For example, an oblivious agent has a single state, or, equivalently, zero bits of *persistent* memory. Note that in this model the size of the agent memory represents the amount of information that the agent can remember *while moving between nodes in the graph*. This does not restrict computations made on a node and thus the transition function can be any deterministic function. Additional memory needed for computations can be seen as provided temporarily by the hosting node. Nevertheless, our agent

algorithms perform very simple tests and operations on the non-constant inputs i and d, namely equality tests and incrementations.

Periodic graph exploration requires that the agent has to visit every node infinitely many times in a periodic manner. In this paper, we are interested in minimising the length of the exploration period. In other words, we want to minimise the maximum number of edge traversals performed by the agent between two consecutive visits of a generic node, while the agent enters this node in the same state through the same port.

Cohen et al. [4] showed that putting two bits of advice at each node allows to explore all graphs by an agent with constant memory, by a periodic traversal of length $O(m)$, where m is the number of edges. In the general adversarial setting (where the adversary can set the port numbers in a misleading order), the exploration problem is unsolvable, even restricted to cubic planar graphs [5]. On the other hand, even if nodes are not marked in any way but if port numbers are carefully assigned (still satisfying the condition that at each node v, port numbers from 1 to d_v are used), then a simple agent, even oblivious, can perform periodic graph exploration within period of length $O(n)$. Using appropriate assignment of the local port numbers, the best known period achieved by an oblivious agent is $10n$ [2] whereas the best known period achieved by an agent with constant memory is $3.75n$ [3].

1.1 Related Work

Graph exploration by robots has recently attracted growing attention. The unknown environment in which the robots operate is often modelled as a graph, assuming that the robots may only move along its edges. The graph setting is available in two different forms.

In [6, 7, 8, 9, 10], the robot explores strongly connected directed graphs and it can move only in one pre-specified direction along each edge. In [11, 12, 4, 13, 14, 15, 16], the explored graph is undirected and the agent can traverse edges in both directions. Also, two alternative efficiency measures are adopted in most papers devoted to graph exploration, namely, the *time* of completing the task [6, 11, 7, 8, 12, 9, 13], or the number of *memory bits* (states in the automaton) available to the agent.

In this paper, we are interested in robots characterised by very low memory utilisation. In fact, the robots are allowed to use only a constant number of memory bits. This restriction permits modelling robots as finite state automata. Budach [1] proved that no finite automaton can explore all graphs. Rollik [5] showed later that even a finite team of finite automata cannot explore all planar cubic graphs. This result is improved in [17], where Cook and Rackoff introduce a powerful tool, called the *JAG*, for Jumping Automaton for Graphs. A JAG is a finite team of finite automata that permanently cooperate and that can use *teleportation* to move from their current location to the location of any other automaton. However, even JAGs cannot explore all graphs [17].

2 Preliminaries

2.1 Notation and Basic Definitions

Let $G = (V, E)$ be a simple, connected, undirected graph. We denote by $\overrightarrow{G}$ the symmetric directed graph obtained from G by replacing each undirected edge $\{u, v\}$ by two directed edges in opposite directions – the directed edge from u to v denoted by (u, v) and the directed edge from v to u denoted by (v, u). For each directed edge (u, v) or (v, u) we say that the undirected edge $\{u, v\} \in G$ is its *underlying* edge. For any node v of a directed graph the *out-degree* of v is the number of directed edges leaving v, the *in-degree* of v is the number of directed edges incoming to v, and the *cumulative degree* of v is the sum of its out-degree and its in-degree.

Directed cycles constructed by our algorithm traverse some edges in G once and some other edges twice in opposite directions. However, at early stages, our algorithm for oblivious agents is solely interested in whether the edge is unidirectional or bidirectional, indifferently of the direction. To alleviate the presentation (despite some abuse of notation), in this context, an edge that is traversed once when deprived of its direction is called a *single edge*. Similarly, an edge that is traversed twice is called a *two-way edge*, and it is understood to be composed of two single edges (in opposite directions). Hence we extend the notion of single and two-way edges to general directed graphs in which the direction of edges is removed. In particular, we say that two remote nodes s and t are connected by a *two-way path*, if there is a finite sequence of vertices $v_1, v_2, \ldots, v_k$, where each pair v_i and v_{i+1} is connected by a two-way edge, and $s = v_1$ and $t = v_k$. We call a directed graph $\overrightarrow{K}$ *two-way connected* if for any pair of nodes there is a two-way path connecting them. Note that two-way connectivity implies strong connectivity but not the opposite.

2.2 Three-Layer Partition

The three-layer partition is a new graph decomposition method that we use in constructing periodic tours efficiently in both the oblivious and the constant-memory cases.

For any set of nodes X we call the *neighborhood* of X the set of their neighbors in graph G (excluding nodes in X) and we denote it by $N_G(X)$. One of the main components of the constructions of our technique are *backbone trees* of G, that is, connected cycle-free subgraphs of G. We say that a node v is *saturated* in a backbone tree T of G if all edges incident to v in G are also present in T.

A *three-layer partition* of a graph $G = (V, E)$ is a 4-tuple (X, Y, Z, T_B) such that (1) the three sets X, Y and Z form a partition of V, (2) $Y = N_G(X)$ and $Z = N_G(Y) \setminus X$, (3) T_B is a tree of node-set $X \cup Y$ where all nodes in X are saturated. We call X the *top layer*, Y the *middle layer*, and Z the *bottom layer* of the partition. Any edge of G between two nodes in Y will be called *horizontal*.

During execution of procedure 3L-Partition the nodes in V are dynamically partitioned into sets X, Y, Z, P and R with temporary contents, where X is

the set of saturated nodes, $Y = N_G(X)$ contains nodes at distance 1 from X, $Z = N_G(Y) \setminus X$ contains nodes at distance 2 from X, $P = N_G(Z) \setminus Y$ contains nodes at distance 3 from X and $R = V \setminus (X \cup Y \cup Z \cup P)$ contains all the remaining nodes in V.

Procedure. 3L-PARTITION($in : G = (V, E)$; $out : X, Y, Z, T_B$);
(1) $X = Y = Z = P = \emptyset$; $R = V$; $T_B = \emptyset$;
(2) select an arbitrary node $v \in R$;
(3) **loop**
  (a) $X = X \cup \{v\}$; (insert into X newly selected node);
  (b) update contents of sets Y, Z, P and R (on the basis of new X);
  (c) saturate the newly inserted node v to X (i.e., insert all new edges in T_B);
  (d) **if** the new node v in X was selected from P **then** insert in T_B an arbitrary horizontal edge (on middle level) to connect the newly formed star rooted in v with the rest of T_B.
  (e) **if** any new node $v \in Y$ can be saturated **then**
    select v for saturation;
    else-if any new node $v \in Z$ can be saturated **then**
      select v for saturation;
      else-if P is non-empty **then**
        select a new v from P for saturation arbitrarily;
        else exit-loop;
  end-loop
(4) **output** (X, Y, Z, T_B)

Figure 1 below shows a representative example of the output from the 3L-PARTITION procedure.

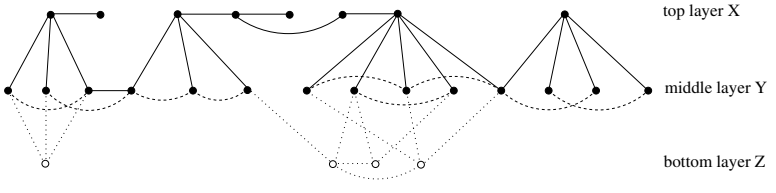


Fig. 1. Three-layer partition. Solid lines and black nodes belong to the backbone tree T_B. Dashed lines represent horizontal edges outside T_B. Dotted lines are incident to nodes from Z.

Lemma 1. *Procedure* 3L-PARTITION *computes a three-layer partition for any connected graph* G.

Lemma 2. *The three-layer partition has the following properties:*
(1) *each node in* Y *has an incident horizontal edge outside of* T_B*;*
(2) *each node in* Z *has at least two neighbors in* Y.

Proof. To prove property (1) assume, by contradiction, that there exists a node $u \in Y$ that has no horizontal edges outside of T_B. Observe that in this case u can be saturated , i.e., u may be moved to X, inserting into T_B all remaining edges incident to u. Indeed, since before u was saturated all such edges lead only to nodes in Z their insertion does not form cycles. Thus property (1) holds. Finally, assume there is a node w in Z with no more than one incident edge leading to level Y. Also in this case we can saturate w since all edges incident to w form a star that shares at most one node with T_B. Thus, no cycle is created, which in turn proves property (2). □

Lemma 3. *For any graph $G = (V, E)$ a three-layer partition may be computed in $O(|E|)$ time.*

2.3 RH-Traversability and Witness Cycles

In this section we discuss the conditions for the oblivious periodic traversals. Given a port number assignment algorithm and an agent algorithm, it is possible, for a given degree d, to permute all port numbers incident to each degree-d node of a graph G according to some fixed permutation σ, and to modify the transition function f of the agent accordingly, so that the agent behaves exactly the same as before in G. The new transition function f' is in this case given by the formula $f' = \sigma \circ f \circ \sigma^{-1}$ and the two agent algorithms are said to be equivalent.

More precisely, two agent algorithms described by their respective transition functions f and f' are *equivalent* if for any $d > 0$ there exists a permutation σ on $\{1, \ldots, d\}$ such that $f' = \sigma \circ f \circ \sigma^{-1}$.

The most common algorithm used for oblivious agents is the Right-Hand-on-the-Wall algorithm. This algorithm is specified by the transition function $f : (s, i, d) \mapsto (s, (i \bmod d)+1)$. Differently speaking, if the agent enters a degree-d node v by port number i, it will exit v through port number $(i \bmod d) + 1$.

The following lemma states that any couple consisting of a port number assignment algorithm and an oblivious agent algorithm, and solving the periodic graph exploration problem, can be expressed by using the Right-Hand-on-the-Wall algorithm as the agent algorithm. We will thus focus on this algorithm in all subsequent parts referring to oblivious agents.

Lemma 4. *Any agent algorithm enabling an oblivious agent to explore all graphs (even all stars) is equivalent to the Right-Hand-on-the-Wall algorithm.*

Graph traversal according to the Right-Hand-on-the-Wall algorithm has been called *right-hand traversals* or shortly *RH-traversals*, see [2]. Similarly, cyclic paths formed in the graph according to the right-hand rule are called *RH-cycles*. The aim of our first oblivious-case algorithm is to find a short RH-traversal of the graph, i.e., to find a cycle $\overrightarrow{C}$ in $\overrightarrow{G}$ containing all nodes of $\overrightarrow{G}$ and satisfying the right-hand rule: If $e_1 = (u, v)$ and $e_2 = (v, w)$ are two successive edges of $\overrightarrow{C}$ then e_2 is the successor of e_1 in the port numbering of v. We call such a cycle a *witness cycle* for G, and the corresponding port numbering a *witness port numbering*.

Given graph $\overrightarrow{G}$ we first design $\overrightarrow{H}$, a spanning subgraph of $\overrightarrow{G}$ that contains all edges of a short witness cycle $\overrightarrow{C}$ of $\overrightarrow{G}$. Then we look for the port numbering of each node in $\overrightarrow{H}$ to obtain $\overrightarrow{C}$. The characterisation of such a graph $\overrightarrow{H}$ is not trivial, however it is easy to characterise graphs which are unions of RH-cycles.

Definition 5. *A node $v \in \overrightarrow{G}$ is* RH-traversable *in $\overrightarrow{H}$ if there exists a port numbering π_v such that, for each edge $(u, v) \in \overrightarrow{H}$ incoming to v via an underlying edge e there exists an outgoing edge $(v, w) \in \overrightarrow{H}$ leaving v via the underlying edge e', such that e' is the successor of e in the port numbering of v.*

We call such ordering a witness ordering *for v.*

Let $\overrightarrow{H}$ be a spanning subgraph of $\overrightarrow{G}$. For each node v, denote by b_v, i_v and o_v the number of two-way edges incident to v used in $\overrightarrow{H}$, only incoming and only outgoing edges, respectively. The following lemma characterises the nodes of a graph being a union of RH-cycles.

Lemma 6. *A node v is RH-traversable if and only if $b_v = d_v$ or $i_v = o_v > 0$.*

Proof. ($\Rightarrow$) The definition of RH-traversability implies $i_v = o_v$.
($\Leftarrow$) If $b_v = d_v$, i.e., all edges incident to v are used in two directions, any ordering of the edges is acceptable. Otherwise ($b_v \neq d_v$,) choose a port numbering in which outgoing edges that contribute to two-way edges are arranged in one block followed by an outgoing edge. All remaining directed edges are placed in a separate block, in which edges alternate directions and the last (incoming) edge precedes the block of all two-way edges. □

We easily obtain the following:

Corollary 7. *A spanning subgraph $\overrightarrow{H}$ of $\overrightarrow{G}$ is a union of RH-cycles if and only if each node v has an even number of single edges incident to v in $\overrightarrow{H}$, and, in case no single edge is incident to v in $\overrightarrow{H}$, all two-way edges incident to v in $\overrightarrow{G}$ must be also present in $\overrightarrow{H}$.*

In the rest of this section we introduce several operations on cycles, and the conditions under which these operations will result in a witness cycle.

Consider a subgraph $\overrightarrow{H}$ of G that has only RH-traversable nodes. Observe that any port numbering implies a partitioning of $\overrightarrow{H}$ into a set of RH-cycles. Take any ordering γ of this set of cycles. We define two rules which transform one set of cycles to another. The first rule, *Merge3*, takes as an input three cycles incident to a node and merges them to form a single one. The second rule, *EatSmall*, breaks a non-simple cycle into two sub-cycles and transfers one of them to another cycle.

1. **Rule Merge3:** Let v be a node incident to at least three different cycles C_1, C_2 and C_3. Let x_1, x_2 and x_3 be the underlying edges at v containing incoming edges for cycles C_1, C_2 and C_3, respectively (x_1, x_2 and x_3 can be

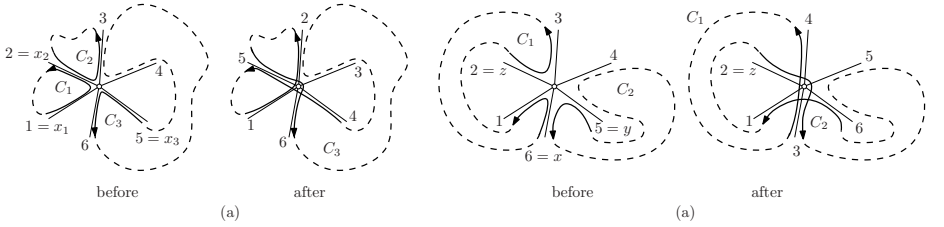


Fig. 2. (a) Applying rule *Merge3*; (b) applying rule *EatSmall*

a single edge or a two-way edge in $\overrightarrow{H}$). Suppose w.l.o.g., that x_2 is between x_1 and x_3 in cyclic port numbering of v. The port numbering which makes the successor of x_2 become the successor of x_1, the successor of x_3 become the successor of x_2 and the successor of x_1 become the successor of x_3 and keeps the relative order of the remaining edges the same (see Figure 2(a)) connects the cycles C_1, C_2 and C_3 into a single cycle C_3, while remaining a witness port numbering for v (due to the original port numbering).

2. **Rule EatSmall:** Let C_1 be the smallest cycle in ordering γ such that
   - there is a node v that appears in C_1 at least twice
   - there is also another cycle C_2 incident to v
   - $\gamma(C_1) < \gamma(C_2)$

   Let x and y be underlying edges at v containing incoming edges for C_1 and C_2, respectively; let z be the underlying edge containing the incoming edge by which C_1 returns to v after leaving via the successor of x. If z is the successor of y, choose a different x. Modify the ordering of the edges in v as follows: (1) the successor of x becomes the new successor of y, (2) the old successor of y becomes the new successor of z, (3) the old successor of z becomes the new successor of x and (4) the order of the other edges remains unchanged – see Figure 2(b).

Lemma 8. *Let $\overrightarrow{K}$ be a two-way connected spanning subgraph of G with all nodes RH-traversable in $\overrightarrow{K}$. Consider the set of RH-cycles generated by some witness port numbering of its nodes, with C^* being the largest cycle according to some ordering γ. If neither* Merge3 *nor* EatSmall *can be applied to the nodes of C^* then C^* is a witness cycle.*

Proof. Suppose, by contradiction, that C^* does not span all the nodes in G. Let V' be the set of nodes of G not traversed by C^*. Since $\overrightarrow{K}$ is two-way connected there exist two nodes $u, v \in G$, such that v belongs to C^* and $u \in V'$, and the directed edges (u, v) and (v, u) belong to $\overrightarrow{K}$. Edges (u, v) and (v, u) cannot belong to different cycles of $\overrightarrow{K}$ because *Merge3* would be applicable. Hence (u, v) and (v, u) must both belong to the same cycle C'. However (u, v) and (v, u) cannot be consecutive edges of C' because this would imply $d_v = 1$ which is not the case, since v also belongs to C^*. Hence C' must visit v at least twice. However, since C^* is the largest cycle we have $\gamma(C') < \gamma(C^*)$ and the conditions

of applicability of rule *EatSmall* are satisfied with $C_1 = C'$ and $C_2 = C^*$. This is the contradiction proving the claim of the lemma. □

3 Oblivious Periodic Traversal

In this section we describe the algorithm that constructs a short witness cycle for graph G. This witness cycle will allow an oblivious agent (i.e., one with no persistent memory) to perform the periodic traversal of G. According to Lemma 8 it is sufficient to construct a spanning subgraph $\overrightarrow{K}$ of G which is two-way connected, such that, each node of G is RH-traversable in $\overrightarrow{K}$. We will present first a restricted case of a *terse set of RH-cycles*, when it is possible to construct a spanning tree of G with no saturated node. In this case we can construct a witness cycle of size $2n$. In the case of arbitrary graphs, we need a more involved argument, which will lead to a witness cycle of size $4\frac{1}{3}n$. We conclude this section with the presentation of a lower bound of $2.8n$.

3.1 Terse Set of RH-Cycles

Suppose that we have a graph G, which has a spanning tree T with no saturated node. This happens for large and non-trivial classes of graphs, including two-connected graphs, graphs admitting two disjoint spanning trees, and many others. For those graphs we present an algorithm that finds a shorter witness cycle than one that we can find for arbitrary graphs. The idea of the algorithm is to first construct a spanning subgraph of G, $\overrightarrow{K}$ of size $2n$, which contains only RH-traversable nodes (cf. algorithm TerseCycles). Then we apply a port numbering which partitions $\overrightarrow{K}$ into a set of RH-cycles that can then be merged into a single witness cycle (cf. Corollary 10).

Algorithm. TerseCycles:

1: Find T – a spanning subgraph of G with no saturated nodes;
2: $\overrightarrow{K} \leftarrow T$; {each edge in T is a two-way edge in $\overrightarrow{K}$}
3: For each node $v \in \overrightarrow{K}$ add to $\overrightarrow{K}$ a single edge from $G \setminus T$; {the single edges form a collection of stars S}
4: Restore-Parity($\overrightarrow{K}, T, root(T)$);

Procedure Restore-Parity has to assure that the number of single edges incident to each node is even. The procedure visits each node v of the tree T in the bottom-up manner and counts all single edges incident to v. If this number is odd, the two-way edge leading to the parent is reduced to a single edge (with the direction to be specified later). The procedure terminates when the parity of all children of the root in the spanning tree is restored. Note also that the cumulative degree of the root must be even since the cumulative degree of all nodes in S is even. Note also that no decision about the direction of single edges is made yet.

Procedure. RESTOREPARITY(directed graph $\overrightarrow{K}$, tree T, node v): integer;

1: $P_v = (\text{number of single edges in } \overrightarrow{K} \setminus T) \pmod 2$;
2: **if** v is not a leaf in T **then**
3: **for** each node $c_v \in T$ being a child of v **do**
4: $P_v \leftarrow (P_v + RestoreParity(\overrightarrow{K}, T, c_v)) \pmod 2$;
5: **end for**
6: **end if**
7: **if** $P_v = 1$ **then**
8: reduce the two-way edge $(P, parent(P))$ to single;
9: **end if**
10: **return** P_v;

Lemma 9. *After the completion of procedure* TERSECYCLES *every node of* $\overrightarrow{K}$ *is RH-traversable.*

Proof. Every node is either saturated or it has at least two single edges incident to it. □

Corollary 10. *For any graph G admitting a spanning tree T, such that none of the nodes is saturated (i.e., $G \setminus T$ spans all nodes of G) it is possible to construct a witness cycle of length at most $2n$.*

Corollary 10 gives small witness cycles for a large class of graphs. It should be noted for 3-regular graphs, finding a spanning tree having no saturated nodes corresponds to finding a Hamiltonian path, a problem known to be NP-hard even in this restricted setting [18].

3.2 Construction of Witness Cycles in Arbitrary Graphs

The construction of witness cycles is based on the following approach. First select a spanning tree T of graph G composed of two-way edges. Let G_i, for $i = 1, 2, \ldots, k$ be the connected components of $G \setminus T$, having, respectively, n_i nodes. For each such component we apply procedure 3L-PARTITION, obtaining three sets X_i, Y_i and Z_i and a backbone tree T_i. We then add single edges incident to the nodes of sets Y_i and Z_i, and we apply the procedure RESTOREPARITY to each component G_i. We do this in such a way that the total number of edges in G_i is smaller than $2\frac{1}{3}n$. For the union of graphs $T \cup G_1 \cup G_2 \cup \cdots \cup G_k$ we take a port numbering that generates a set of cycles. The port numbering and orientation of edges in the union of graphs is obtained as follows. First we remove temporarily all two-way edges from the union. The remaining set of single edges is partitioned into a collection of simple cycles, where edges in each cycle have a consistent orientation. Further we reinstate all two-way edges in the union, such that each two-way edge is now represented as two arcs with the opposite direction. Finally we provide port numbers at each node of the union, such that it is consistent with the RH-traversability condition, see Lemma 6. We apply rules *Merge3* and *EatSmall* to this set of cycles until neither rule can be applied. The set of cycles obtained will contain a witness cycle, using Lemma 8.

Algorithm. FindWitnessCycle;

1: Find a spanning tree T of graph G {two-way edges}
2: **for** each connected component G_i of $G \setminus T$ **do**
3: 3L-Partition$(G_i, X_i, Y_i, Z_i, T_i)$;
4: Form set P_i by selecting for each node in Z_i two edges leading to Y_i; {single edges};
5: Form a set of independent stars S_i spanning all nodes in Y_i that are not incident to P_i; {single edges};
6: RestoreParity$(G_i \cup P_i \cup S_i, T_i, root(T_i))$;
7: **end for**
8: $\overrightarrow{K} \leftarrow T \cup G_1 \cup G_2 \cup \cdots \cup G_k$;
9: Take any port numbering and produce a set $\mathfrak{C}$ of RH-cycles induced by it;
10: Apply repeatedly *Merge3* or, if not possible, *EatSmall* to $\mathfrak{C}$ until neither rule can be applied;
11: **return** the witness cycle of $\mathfrak{C}$;

Theorem 11. *For any n-node graph algorithm* FindWitnessCycle *returns a witness cycle of size at most $4\frac{1}{3}n - 4$.*

Theorem 12. *The algorithm* FindWitnessCycle *terminates in $O(|E|)$ time.*

3.3 Lower Bound

We have shown in the previous section that for any n-node graph we can construct a witness cycle of length at most $4\frac{1}{3}n - 4$. In this section we complement this result with the lower bound $2.8n$:

Theorem 13. *For any non-negative integers n, k and l such that, $n = 5k + l$ and $l < 5$, there exists an n-node graph for which any witness cycle is of length $14k + 2l$.*

Proof. Consider first a single *diamond graph* G', see the left part of Figure 3. Without loss of generality, we can assume that we start the traversal through (v, x). Consider the successor of (x, u). Also, without loss of generality, we can take (u, y) as the successor. Now there is only one feasible successor of (y, v) and that is (v, z). All other edges violate either RH-traversability ((v, y)) or leave z unvisited. Similarly, the only possible successor of (z, u) is (u, x) ((u, y) has already been traversed with a different predecessor, and (u, z) violates RH-traversability), of (x, v) is (v, y) and of (y, u) is (u, z). Therefore, each edge of G' must be used in both directions.

Consider now a chain of diamond graphs from the right side of Figure 3, starting the graph traversal at node v_0. From the fact that each edge in the witness cycle is traversed at most twice (one time in each direction) it follows that when returning from v_i to u_{i-1}, all nodes in G_i (as well as in all G_j, for $j > i$) must have been visited. Note that from RH-traversability it follows that the successor of (u_{i-1}, v_i) cannot be the same (in reverse direction) as the predecessor of (v_i, u_{i-1}), and similarly the successor of (v_i, u_{i-1}) cannot be the

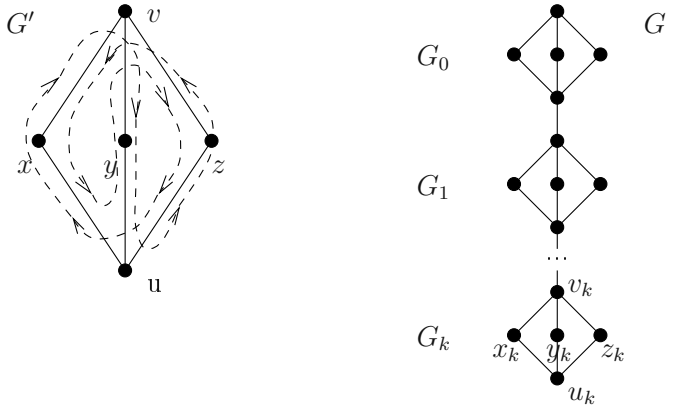


Fig. 3. The lower bound based on diamond graphs

same as the predecessor of (u_{i-1}, v_i). In turn this means that the analogous arguments (as used in G') apply also to each G_i, therefore all edges of G must be traversed in both directions.

The theorem now follows directly for $n = 5k$. If n is not a multiple of 5, an extra path of l nodes can be added to u_k to satisfy the claim of the theorem. □

4 Periodic Traversal with Constant Memory

In this section we focus on the construction of a tour in arbitrary undirected graphs to be traversed by an agent equipped with a constant memory. The use of the constant amount of memory allows the agent to change its behavior between a small number of (internal) states for its operation, i.e., the agent has a deterministic transition function and can change from one state to another according to pre-defined rules. As in the case of oblivious agents, we do not impose restrictions on the amount of *local* memory it might have available for use at any vertex, but this local memory is temporary and is lost when an agent leaves the vertex. The main idea of the periodic graph traversal mechanism proposed in [19], and further developed in [3], is to visit all nodes in the graph while traversing along an *Euler tour* of a (particularly chosen) spanning tree (together with a few additional, specially chosen, edges). Due to space constraints, we refer the reader to [3] for more background and details on the mechanism the agent uses to perform the exploration. In what follows, we concentrate on the new construction of the spanning tree (with additional edges) that the agent uses for its exploration.

Recall that the nodes of the input graph can be partitioned into three sets X, Y and Z where all nodes in X and Y are spanned by a backbone tree, see Section 2.2. The spanning tree T is obtained from the backbone tree by connecting every node in Z to one of its neighbors in Y. Recall also that every node $v \in X$ is *saturated*, i.e., all edges incident to v in G belong also to the spanning

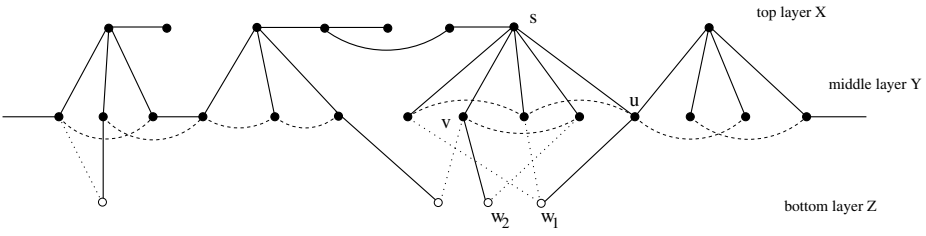


Fig. 4. Fragment of the spanning tree with the root located to the right of w_1 and w_2

tree. Every node in Y that lies on a path in T between two nodes in X is called a *bonding node*. The remaining nodes in Y are called *local*.

Initial port labeling. When the spanning tree T is formed, we pick one of its leaves as the root r where the two ports located on the tree edge incident to r are set to 1. Initially, for any node v the port leading to the parent is set to 1 and ports leading to the i children of v are set to $2, \ldots, i+1$, such that the subtree of v rooted in child j is at least as large as the subtree rooted in child j', for all $2 \leq j < j' \leq i+1$. All other ports are set arbitrarily using distinct values from the range $i+2, \ldots, d_v$, where d_v is the degree of v. Later, we modify the allocation of ports at certain leaves of the spanning tree located in Z. In particular we change labels at all children having no other leaf-siblings in T of bonding nodes (see, e.g., node w_1 in Figure 4), as well as at single children of local nodes, but only if the local node is the last child of a node in X that has children on its own (see, e.g., node w_2 in Figure 4).

Port swap operation. Recall that every leaf w located at the level Z has also an incident edge e outside of T that leads to some node v in Y (property 2 of the three-layer partition). When we swap port numbers at w, we set to 2 the port on the tree edge leading to the parent of w. We call such edge a *sham penalty edge* since it now pretends to be a penalty edge while, in fact, it connects w to its parent in the spanning tree T. We also set to 1 the port number on the lower end of e. All other port numbers at w (if there are more incident edges to w) are set arbitrarily. After the port swap operation at w is accomplished we also have to ensure that the edge e will never be examined by the agent, otherwise it would be wrongly interpreted as a legal tree edge, where v would be recognised as the parent of w. In order to avoid this problem we also set ports at v with greater care. Note that v has also an incident horizontal edge e' outside of T (property 1 of the three-layer partition). Assume that the node v has i children in T. Thus if we set to $i+2$ the port on e' (recall that port 1 leads to the parent of v and ports $2, .., i+1$ lead to its children) the port on e will have value larger than $i+2$ and e will never be accessed by the agent. Finally note that the agent may wake up in the node with a sham penalty edge incident to it. For this reason we introduce an extra state to the finite state automaton $\mathcal{A}$ governing moves of the agent in [3] to form a new automaton $\mathcal{A}^+$. While being in the wake up state the agent moves across the edge accessible via port 1 in order to start regular

performance (specified in [3]) in a node that is not incident to the lower end of a sham penalty edge.

Lemma 14. *The new port labeling provides a mechanism to visit all nodes in the graph in a periodic manner by the agent equipped with a finite state automaton* $\mathcal{A}^+$.

Theorem 15. *For any undirected graph* G *with* n *nodes, it is possible to compute a port labeling such that an agent equipped with a finite state automaton* $\mathcal{A}^+$ *can visit all nodes in* G *in a periodic manner with a tour length that is no longer than* $3\frac{1}{2}n - 2$.

Note that in the model with implicit labels, one port at each node has to be distinguished in order to break symmetry in a periodic order of ports. This is to take advantage of the extra memory provided to the agent.

5 Conclusion

Further studies on trade-offs between the length of the periodic tour and the memory of a mobile entity are needed. The only known lower bound $2n - 2$ holds independently of the size of the available memory, and it refers to trees. This still leaves a substantial gap in view of our new $3.5n$ upper bound. Another alternative would be to look for as good as possible tour for a given graph, for example, in a form of an approximate solution. Indeed, for an arbitrary graph, finding the shortest tour may correspond to discovering a Hamiltonian cycle in the graph, which is NP-hard.

Acknowledgements. Many thanks go to Adrian Kosowski, Rastislav Kralovic, and Alfredo Navarra for a number of valuable discussions on the main themes of this work.

References

[1] Budach, L.: Automata and labyrinths. Mathematische Nachrichten, 195–282 (1978)
[2] Dobrev, S., Jansson, J., Sadakane, K., Sung, W.K.: Finding short right-hand-on-the-wall walks in graphs. In: Pelc, A., Raynal, M. (eds.) SIROCCO 2005. LNCS, vol. 3499, pp. 127–139. Springer, Heidelberg (2005)
[3] Gąsieniec, L., Klasing, R., Martin, R., Navarra, A., Zhang, X.: Fast periodic graph exploration with constant memory. J. Computer and System Science 74(5), 808–822 (2008)
[4] Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. ACM Transactions on Algorithms 4(4), 331–344 (2008)
[5] Rollik, H.: Automaten in planaren graphen. Acta Informatica 13, 287–298 (1980)
[6] Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM J. Computing 29, 1164–1188 (2000)

[7] Bender, M., Fernandez, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: Exploring and mapping directed graphs. Information and Computation 176(1), 1–21 (2002)
[8] Bender, M., Slonim, D.K.: The power of team exploration: two robots can learn unlabeled directed graphs. In: Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS), pp. 75–85 (1994)
[9] Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. J. Graph Theory 32(3), 265–297 (1999)
[10] Fleischer, R., Trippen, G.: Exploring an unknown graph efficiently. In: Proc. 13th Annual European Symposium on Algorithms (ESA), pp. 11–22 (2005)
[11] Awerbuch, B., Betke, M., Rivest, R., Singh, M.: Piecemeal graph exploration by a mobile robot. Information and Computation 152(2), 155–172 (1999)
[12] Betke, M., Rivest, R., Singh, M.: Piecemeal learning of an unknown environment. Machine Learning 18(2-3), 231–254 (1995)
[13] Duncan, C., Kobourov, S., Kumar, V.: Optimal constrained graph exploration. ACM Transaction on Algorithms 2(3), 380–402 (2006)
[14] Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theoretical Computer Science 345(2-3), 331–344 (2005)
[15] Fraigniaud, P., Ilcinkas, D., Rajsbaum, S., Tixeuil, S.: The reduced automata technique for graph exploration space lower bounds. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) Theoretical Computer Science. LNCS, vol. 3895, pp. 1–26. Springer, Heidelberg (2006)
[16] Panaite, P., Pelc, A.: Exploring unknown undirected graphs. J. Algorithms 33, 281–295 (1999)
[17] Cook, S.A., Rackoff, C.: Space lower bounds for maze threadability on restricted machines. SIAM J. Computing 9(3), 636–652 (1980)
[18] Garey, M., Johnson, D., Tarjan, R.: The planar Hamiltonian circuit problem is NP-complete. SIAM J. Computing 5(4), 704–714 (1976)
[19] Ilcinkas, D.: Setting port numbers for fast graph exploration. Theoretical Computer Science 401, 236–242 (2008)

Black Hole Search in Directed Graphs*

Jurek Czyzowicz[1], Stefan Dobrev[2], Rastislav Královič[3], Stanislav Miklík[3], and Dana Pardubská[3]

[1] Université du Québec en Outaouais
Gatineau, Québec, Canada
Jurek.Czyzowicz@uqo.ca
[2] Slovak Academy of Sciences
Bratislava, Slovakia
stefan.dobrev@savba.sk
[3] Comenius University
Bratislava, Slovakia
{kralovic,miklik,pardubska}@dcs.fmph.uniba.sk

Abstract. We consider the problem of cooperative network exploration by agents under the assumption that there is a harmful host present in the network that destroys the incoming agents without outside trace – the so-called *black hole search* problem.

Many variants of this problem have been studied, with various assumptions about the timing, agents' knowledge about the topology, means of inter-agent communication, amount of writable memory in vertices, and other parameters. However, all this research considered undirected graphs only, and relied to some extent on the ability of an agent to mark an edge as safe immediately after having traversed it.

In this paper we study directed graphs where this technique does not apply, and show that the consequence is an exponential gap: While in undirected graphs $\Delta+1$ agents are always sufficient, in the directed case at least 2^{Δ} agents are needed in the worst case, where Δ is in-degree of the black hole. This lower bound holds also in the case of synchronous agents. Furthermore, we ask the question *What structural information is sufficient to close this gap?* and show that in planar graphs with a planar embedding known to the agents, $2\Delta+1$ agents are sufficient, and 2Δ agents are necessary.

1 Introduction

Consider a set of mobile *agents* roaming in an environment represented by a graph. In the *graph exploration problems* the common goal of the agents is to extract some unknown information about the graph by collecting local information from particular vertices and communicating the partial results among themselves. Perhaps the oldest problem from this class is the problem of graph exploration by a finite automaton (see e.g. [7,8,20,25] and references therein) where a single agent is modeled by a finite state machine, and the goal is to

* Research supported by grant APVV-0433-06.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 182–194, 2010.

traverse the whole graph. Numerous other problems and variants have been considered for teams of agents, often with the requirement to minimize the agents' memory (e.g. [1,2,3,4,10,18,21,22,23,28,30]).

More recently, graph exploration problems in *faulty* networks came into consideration, motivated partially by the software-engineering paradigm of mobile computations [26], where pieces of code belonging to a user are transmitted over the network and executed on other hosts. This approach has been proven efficient from the point of view of network performance, however there are many issues (also involving security aspects) to be addressed [17,24,29,31]. One of them is the existence of *malicious hosts* that do not execute agents properly but try to use them to actually harm the agents' owner.

Graph exploration problems in faulty networks introduce an adversary of various kinds to the network that makes the collective effort of the agents harder. The best understood problem in this family is the so called *black hole search* problem [9,11,12,13,14,15,16,27], in which the network contains one (or more) malicious hosts called black holes, which destroy incoming agents without any trace. The goal of the agents is to locate the position(s) of the black hole(s), or at least to find a *safe spanner*: a connected subgraph containing all vertices except black holes.

The black hole search problem has been studied in numerous variants with agents of different capabilities, and on different networks. The main task is to understand how the interplay between the network structure, and agent's power and knowledge influence the solvability and complexity of the problem. In particular, the models include synchronous, semi-synchronous or asynchronous agents; with different means of communication (face-to-face, white-board, pebbles); anonymous or non-anonymous; with complete map of the network, partial topological information, or in unknown network, etc. The main complexity measure is the number of agents necessary to locate the black hole, and a secondary measure is the overall number of moves in order the reach the goal.

While there are results about exploration of directed graphs (e.g. [4,5,6,19]), to the best of our knowledge all previous results concerning the exploration problems in faulty graphs in general, and the black hole search problem in particular, have considered undirected graphs only. The technique of *cautious walk* [12,15] has been frequently relied upon in order to minimize the number of agents entering the black hole: If an agent wants to traverse an edge, it first marks the edge as *dangerous*. No other agent enters a *dangerous* edge, thus ensuring that at most Δ agents enter the black hole, where Δ is the degree of the black hole. If the exploring agent realizes that the endpoint of the edge is safe, it returns back and marks the edge as safe, allowing other agents to traverse it.

In this paper we consider the black hole search problem in *directed* graphs. In this setting the cautious walk technique can not be applied, and some agents must enter potentially dangerous arcs. In fact, we show that the number of such agents can be quite high and as a consequence 2^{Δ} agents are needed in the worst case in order to locate the black hole. The next question we tackle is: *What kind of structural information is sufficient to close this exponential gap*

between the directed and undirected case? We show that in planar graphs with a planar embedding known by the agents, $2\Delta + 1$ agents are sufficient to locate the black hole, and 2Δ agents are needed in the worst case.

The structure of the paper is as follows: In Section 2 we specify the model and introduce the problem. Section 3 contains the lower bound and related discussion for general directed networks, while in Section 4 the upper and lower bounds for the planar graphs with given planar embedding are presented. In the conclusion Section 5 we discuss related open problems and future research directions.

2 Model and Problem Statement

2.1 Framework

The network is modeled as a simple strongly connected directed graph G consisting of n nodes. The vertices are endowed with *local orientation* – each node can distinguish incident edges (communication ports) by means of locally distinct port labels. No assumption is made about global consistency of these port labels.

Operating in this network is a set of mobile agents. The agents can move from node to a neighbouring node in G, have computing capabilities and bounded storage (polynomial in n). All agents start at the same node, called *home base*. The agents are asynchronous – their computation and movement steps take unpredictable, but finite time. Each node has a bounded amount of storage (polynomial in n) called *white-board*. The agents communicate only by reading and writing on the white-boards; access to a white-board is gained fairly in mutual exclusion. We assume the agents have distinct IDs; notice however that, since the white-boards are accessed in mutual exclusion and the agents start from the same home base, distinct IDs can be easily constructed (by having a counter at the home-base; each agent upon wake-up takes the counter's value as its ID and increments the counter).

2.2 Black Hole Search

One node of the network is a black hole (BH). Any agent that enters the black hole is destroyed; no observable trace of such destruction is available to the other agents. We denote the in-degree of the black hole as Δ.

The black hole search problem is to find the location of the black hole. At least one agents has to survive and terminate. In the moment of termination, each edge is marked as either "safe" or "dangerous"; we require that the "safe" arcs induce a strongly connected spanning subgraph of $G - \{BH\}$. Furthermore, eventually all arcs not leading to the black hole are marked as "safe" (weak termination).

In order to ensure that the black hole search is solvable, we assume that $G - \{BH\}$ is strongly connected, and n is known by the agents (otherwise the agents cannot explore the whole graph, or even if they can, they do not know they did it and it is time to terminate).

The complexity measure we are interested in is the number of agents needed/ sufficient to locate the black hole.

3 Arbitrary Graphs

The agents exploring a directed graph are in a significantly different situation from the undirected case. Since the techniques based on cautious walk cannot be used, agents may be left in a situation when they have to enter a potentially dangerous arc without the option to wait for potential survivors to mark the arc as safe. The following theorem shows the fatal consequences of this forced choice: If Δ arcs lead to the black hole, as many as $2^{\Delta} - 1$ agents may be killed in a directed graph. This is in contrast to the undirected case, where $\Delta + 1$ agents are always sufficient [12].

Theorem 1. *Consider an algorithm $\mathcal{A}$ for locating a black hole in arbitrary directed graphs. For any $\Delta \geq 1$, and any $n > \Delta$ there exists an n-vertex graph G with a black hole of in-degree Δ such that there is an execution of $\mathcal{A}$ on G in which at least 2^{Δ} agents are needed for $\mathcal{A}$ to locate the black hole.*

Proof: For given $\Delta \geq 1$, and $n > \Delta$, consider the family $\mathcal{G}_{\Delta,n}$ of graphs consisting of a directed cycle of $n-1$ vertices $1, 2, \ldots, n-1$, and another vertex 0 with additional arcs $(1,0), (2,0), \ldots, (\Delta,0)$. All outgoing arcs are labelled either 1 or 2 (see Figure 1 for an example).

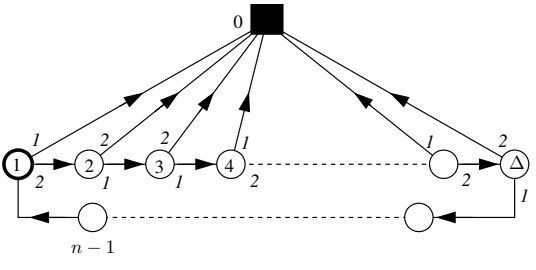


Fig. 1. An example graph from $\mathcal{G}_{\Delta,n}$. Vertex 0 is the black hole.

Let 0 be the black hole, and 1 be the home base. Consider any algorithm $\mathcal{A}$ that locates the black hole on all graphs from $\mathcal{G}_{\Delta,n}$. In order to do so, an agent must return to the home base. In the following analysis we show that 2^{Δ} agents are necessary to ensure this.

First, we prove the following claim using contradiction: *There is a graph $G \in \mathcal{G}_{\Delta,n}$, and an execution of $\mathcal{A}$ on G such that for each vertex $1 \leq i \leq \Delta$ at least half of the agents leaving from i go to the black hole.* Let us assume the claim does not hold, and let $k < \Delta$ be the maximal number for which there is a graph $G \in \mathcal{G}_{\Delta,n}$, and an execution of $\mathcal{A}$ on G such that for each vertex $1 \leq i \leq k$ the claim holds. As we stop the analysis the moment at the first agent returns

to the home base, all agents leaving vertex $k \oplus 1$[1] 1 are distinct. By assumption the majority of them goes to $k \oplus 2$. Let G' be obtained from G by flipping the labels on arcs outgoing from $k \oplus 1$; as G and G' are undistinguishable to $\mathcal{A}$, in G' the majority of agents from $k \oplus 1$ goes to black hole – contradiction with the assumption that k is the largest number for which the claim holds.

Hence (by backwards induction), in order for one agent to leave vertex Δ, there must be at least $2^{\Delta-i}$ distinct agents crossing the arc from i to $i+1$; in particular at least $2^{\Delta-1}$ agents must leave from vertex 1 to vertex 2 (or back to 1 is Δ equals 1). Since at least half of the agents leaving vertex 1 go to the black hole, the necessary number of agents at 1 is at least 2^{Δ}. □

Note that the above arguments apply also in the case the agents are synchronous.

The lower bound suggests the following approach for the upper bound:

Each vertex v maintains on each of its outgoing arcs a counter counting the number of agents that left it via this arc. An agent arriving at v departs via the arc with the lowest counter.

As the graph is strongly connected after removing the black hole, each vertex with an arc to the black hole has also at least one other incident arc and therefore at most half of the agents that enter it will enter at the black hole.

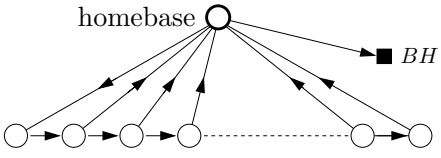


Fig. 2. In the naive algorithm, $\Omega(2^n)$ agents can enter the black hole even for $\Delta = 1$

Unfortunately, this simple idea does not lead to an efficient algorithm, as the agents can return to the same vertex and have more chances to take the arc into the black hole (see Figure 2) – in fact, even if $\Delta = 1$, using this algorithm $O(2^n)$ agents will enter the black hole. It is possible to analyze (and somewhat enhance) this algorithm for arbitrary networks and Δ, however such approach is unsatisfactory as the number of agents needed depends on n, not on Δ.

Intuitively, agents can not die in parts of the graph that are not connected to the black hole; with sufficiently clever algorithm, the size of these parts should not influence the number of agents needed. Hence, our conjecture is:

Conjecture 1. *The number of agents sufficient to locate the black hole depends only on the in-degree of the black hole, and not on the size of the graph.*

4 Planar Graphs

In the previous section we showed that, in general, exponentially more agents may enter the black hole in directed graphs than in undirected. It is therefore

[1] We use dontation $a \oplus i$ to mean i-th vertex from a along the cycle.

a natural question to ask whether there is some topological information that, if given to the agents, improves their survival rate. In this section we show that for planar directed graphs knowing a planar embedding is sufficient to reduce the number of destroyed agents to $O(\Delta)$. Note that since the exponential lower bound from previous section uses planar graphs, the knowledge of the planar embedding is crucial for reducing the number of agents needed.

We consider directed planar graphs with the planar embedding encoded in the arc-labels as follows: in each vertex, the incident arcs (both incoming and outgoing) are labeled clockwise with consecutive integers. Moreover, for each incident arc there is a flag distinguishing whether the arc is outgoing or incoming.

For the upper bound we assume the agents have distinct IDs (recall that those can be assigned in the home base using the mutual exclusion access to the whiteboard there). Analogously, unique vertex ID's can be given to vertices as well: The first agent A(with ID i_A and carrying a counter c_A) entering a vertex set the ID of that vertex to be the pair (i_A, c_A) and increments the counter c_A.

Theorem 2. *There is an algorithm that locates the black hole in planar digraphs with known planar embedding using* $2\Delta + 1$ *agents.*

The basic structure of the algorithm is to maintain a *safe area* – a strongly connected subgraph including the homebase and not containing the black hole. At the beginning of the algorithm the safe area consists of the homebase only. Subsequently, agents traverse parts of the graph, and upon return to the safe area add the newly traversed parts to it. Since the access to the whiteboard in any vertex (and, in particular, in homebase) is guarded by a fair mutex, it is possible to implement all changes to the safe area atomically: an agent which is about to change any information in the safe area first waits to get access to the homebase, then sets a flag there preventing any other agent from concurrently changing the safe area. For the rest of the arguments, when speaking about a state of the safe area, we shall not consider intermediate states when an agent is updating the information.

Agents may be in three states: *exploring* some parts of the graph outside safe area, *waiting* in the homebase, or *queueing* to get the access to the homebase after return from an exploration. However, the latter is for all purposes identical to still traversing the corresponding arcs.

In the exploration phase agents need to store some *private* data in the visited vertices. These data is always signed with agent's ID, so parallel explorations of multiple agents do not interfere. Apart from that, some *public* data may be stored in the vertices: each vertex in the safe area is marked as such, with an arc pointing towards the homebase. Moreover, there is a map of the safe area in the home-base and a list of outgoing arcs from the homebase together with information about which agents traversed them.

The exploration itself is performed by means of *leftmost* and *rightmost traversals*. Intuitively, these correspond to walks in a dungeon with the left/right hand always touching the wall (the arcs correspond to one-way tunnels, and vertices correspond to halls with the outgoing tunnel entrances cyclically ordered according to the planar embedding) until returning back to the safe area.

The main idea of the algorithm is captured in the following lemma:

Lemma 1. *Let A be the safe area and e be an arc outgoing from A. If e does not lead to the black hole then either the leftmost or the rightmost traversal along e reaches A without entering the black hole.*

Proof: Consider an arc e not leading into the black hole. As $G-\{BH\}$ is strongly connected, there must be a path π leading back to A. This path together with A divides the plane into two parts. Suppose w.l.o.g. that the black hole is in the right-hand part.

The leftmost traversal starting along e cannot enter the black hole without first entering A. Otherwise, it would have to cross π, which would contradict the fact that it is a leftmost traversal. □

The leftmost and rightmost traversals are implemented as follows: During the traversal, the agent marks (in its private area) vertices it visited. Moreover, in each visited vertex, there is a list of arcs used by this traversal and, particularly, the most recently used arc is distinguished. The detailed description of the leftmost traversal's implementation follows:
The rightmost traversal is implemented analogously.

The exploring agent performs a left- or rightmost traversal until the safe area is reached (or the agent enters the black hole) with the following exception:

- If, in a vertex v, the traversal would require to traverse an arc already traversed by an agent performing the same (leftmost/rightmost) type of traversal, then wait until the vertex becomes part of the safe area, and then act as if entering the safe area.

Let us call a vertex waiting according to this rule a *blocked* vertex. Note that this rule ensures that at most two agents (one performing leftmost traversal and one performing rightmost traversal) can enter the black hole via any given arc.

Algorithm 1. Leftmost traversal along arc e from a node u

```
 1: mark e as "last used" arc
 2: traverse e, let v be the vertex on the other side of e
 3: if v is marked as visited then
 4:     traverse the cycle formed by the arcs marked as "last used", let v' be the last
            vertex having an unmarked outgoing arc
 5:     go to v', let e' be the last used outgoing arc, let g be the (clockwise) next
            outgoing unmarked arc
 6:     mark e' as "used"
 7:     do leftmost_traversal along g from v'
 8: else
 9:     mark v as visited
10:     let g be the first outgoing arc (clockwise) after e
11:     do leftmost_traversal along g from v
12: end if
```

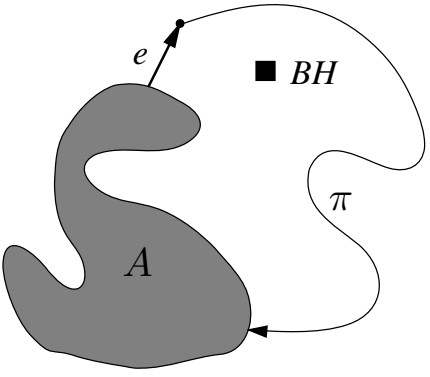


Fig. 3. Leftmost traversal from e cannot enter the black hole

In particular, since the algorithm uses $2\Delta + 1$ agents, it is guaranteed that not all agents enter the black hole and therefore the only way the algorithm could possibly fail is by entering a deadlock.

We shall call an arc that has never been traversed by an agent *unexplored*, an arc that has been traversed by one agent *partially explored*, and an arc traversed by two agents *explored*.

The overall algorithm of an agent is as follows:

Algorithm 2. Algorithm of an agent

1: **while** *safe area has less than* $n - 1$ *vertices* **do**
2: wait for an unexplored or partially explored arc e from (some safe) vertex v to be available; if there are more available arcs, use the priority rules below to choose one
3: if e is unexplored then let *direction* be `left`, else let *direction* be the direction of the traversal that has not yet been performed along e
4: update the list of traversed arcs in the homebase
5: go to v, perform the *direction* `traversal` along e until the safe area is reached again
6: **end while**

The priority rules for selecting an outgoing arc are as follows:

1. If returning from an exploration that started over some partially explored arc e, let X be the other agent that traversed e. (Obviously, X used the opposite-direction traversal. Note that it may be the case that no such agent exists.) If there is a partially explored arc traversed by X outgoing from the currently safe area, choose this arc. If an agent selects an arc according to this rule, we say that it has *paired* with the agent X.
2. Otherwise, if there is any partially explored arc outgoing from the safe area, select one.
3. Else select an unexplored arc.

Lemma 2. *Any agent performing a leftmost (rightmost) traversal that does not lead to the black hole eventually reaches the safe area.*

Proof: Consider an agent A_1 performing a leftmost traversal (the argument for rightmost traversal is analogous) such that A_1 does not reach the safe area. As A_1 will not enter the black hole, the only possibility preventing it from reaching the safe area is if it is blocked, due to an agent A_2 performing leftmost traversal that has already traversed the next arc e on A_1's path. Since the rest of the traversal is uniquely defined by the arc e, A_2 will not enter the black hole either. Using induction on the length of the traversal it can be argued that A_2 eventually reaches the safe area, and v becomes a part of it. Hence, A_1 will eventually reach the safe area, too. □

Now we are ready to finish the proof of the main theorem:
Proof: (of Theorem 2) Each iteration of the loop of the main algorithm increases the safe area, and from the algorithm it follows that at most 2Δ agents enter the black hole. Hence it is sufficient to prove that no agent waits forever.

Consider, for the sake of contradiction, a deadlocked situation. There are two reasons why an agent may be waiting: either it is blocked, or it is waiting in the homebase because no arcs are available for exploration.

First consider the situation with only waiting agents, i.e. each agent has either entered the black hole or is waiting in the homebase. Since no arc is available, all arcs outgoing from the safe area have been traversed by two agents. Moreover, since the graph with the black hole removed is strongly connected, and the algorithm has not finished yet, at least one outgoing arc e does not lead to the black hole. By Lemma 1, at least one of the traversals along e does no lead to black hole, and by Lemma 2 the corresponding agent would have reached the safe area, making e part of the safe area, not an outgoing arc – contradiction.

Let us now consider the situation where at least one agent A is blocked, i.e. is waiting due to the next edge of its traversal having been traversed by an agent B doing traversal in the same direction. By Lemma 2 A's (and B's) traversal must lead to the black hole, otherwise A (B) would eventually unblock and reach the safe area.

Consider, without loss of generality, that first B left the safe area along arc e_1, and later A left the (possibly expanded) safe area via arc e_2. Let e_1' be the arc of B's traversal that was outgoing from the safe area at the moment A left; the situation is as on Figure 4. First note that either e_1' or e_2 is explored in the final configuration: indeed, if neither is explored, then at the moment A decided on the arc to leave the safe area, e_1' was partially explored and e_2 was unexplored – contradicting the rule to prefer partially explored arc to an unexplored one.

Again, without loss of generality let us suppose that e_2 is the explored one, and let X be the other agent having traversed it. Since e_2 does not end in the black hole and A's traversal does lead to the black hole, by Lemma 1 X's traversal does not lead to the black hole. Therefore, by Lemma 2 X returned to the safe area at some time t. At that time, there was an arc e'' from A's traversal outgoing from the safe area (which again might have expanded). Due to the priority rules, if e'' was not explored in time t, X selected it (and hence

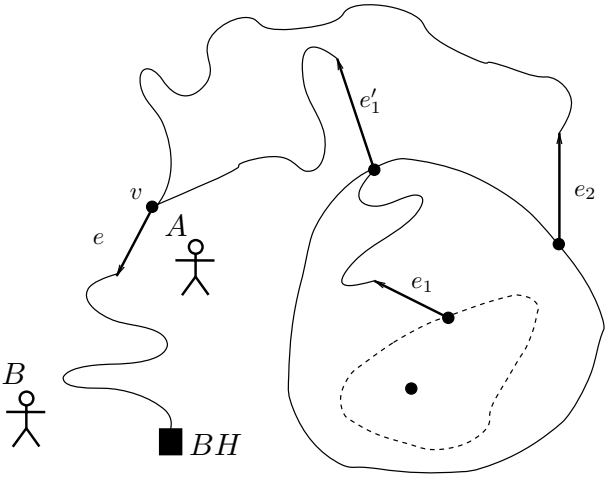


Fig. 4. Situation with blocked agents: First, agent B left the safe area along e_1, then later A left the (increased) safe area via e_2, and e'_1 was outgoing from the safe area at that time. B had in the meantime entered the black hole, and A is waiting in v.

made it explored). In any case, we can repeat the argument with e'' taking the role of e_2, and finally argue by induction that v is a part of the safe area in the final configuration. However, then A is not blocked in v. □

The above algorithm does no rely on the knowledge of Δ, and the knowledge of n is used only to evaluate the termination criteria. With Δ known to the agents and some proper care, the algorithm can be improved to using 2Δ agents. However, it cannot be improved further as the following theorem shows:

Theorem 3. *Consider an algorithm $\mathcal{A}$ that locates a black hole in directed planar graphs. For any Δ there exists a graph G with a black hole of in-degree Δ such that there is an execution of $\mathcal{A}$ on G in which at least $2\Delta - 1$ agents enter the black hole.*

Proof: For any given $\Delta \geq 1$, consider the following class of embedded planar graphs $\mathcal{G}_\Delta$: a $G \in \mathcal{G}_\Delta$ has a distinguished vertex t, and contains a directed cycle s of length $2\Delta+1$. Each vertex of s has one incident arc, in addition to those from the cycle, such that the clockwise sequence of these arcs is $e_0, e_1, \ldots, e_{2\Delta}$ where arcs with odd indices are outgoing and arcs with even indices are incoming. Outgoing arcs e_{2i+1} lead either to t or to an intermediate vertex v_i. Each v_i has two outgoing arcs: left, and right. One of them leads to t, and the other one returns back to s via one of the links e_{2i-1}, e_{2i+1} in such a way as to maintain planarity. If both the right link of v_i and the left link of v_{i+1} return to s via e_{2i+1}, they are connected via a vertex with in-degree two and out-degree one. All arcs e_{2i} that would otherwise be left disconnected are leading from vertex t. See Figure 5 for an example of a graph G.

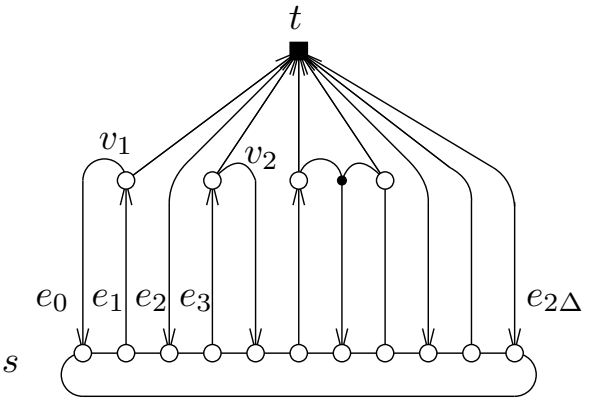


Fig. 5. An example graph $G \in \mathcal{G}_\Delta$. The direction of arcs comprising s is omitted.

Consider the scenario where the homebase of the agents is in s, and t is the black hole. Indeed, any graph $G \in \mathcal{G}_\Delta$ is strongly connected after removing t. Further consider an adversary that delays the agents on links e_i as long as possible (i.e. while there are agents travelling along s). There must be a situation in which there are no more agents travelling in s, and only active agents are on links e_i. The situation percieved by the agents at this point is the same for all graphs from $\mathcal{G}_\Delta$: the sequence of incoming and outgoing arcs. Hence, before some agents return, any algorithm must behave identically on all graphs from $\mathcal{G}_\Delta$ with one exception: the graph with no v-vertices has $2\Delta+2$ vertices; since agents know n, Δ, the algorithm finishes as soon as an agent returns to homebase. In the sequel, we exclude this graph from consideration.

Any algorithm $\mathcal{A}$ starts by a number K of agents (that is the same for all graphs from $\mathcal{G}_\Delta$) leaving s, while other agents may remain waiting for the return of some of them. We prove that if $K < 2\Delta$ then there is a graph $G \in \mathcal{G}_\Delta$ such that all the exploring agents enter the black hole.

The graph is constructed as follows: first consider all arcs e_{2i+1} with at least two agents traversing them, and connect them directly to t. The remaining outgoing arcs lead to v-vertices. Since at most one agent arrives to each v-vertex it is possible to choose the orientation of the v-vertex's outgoing arcs in such a way that the agent enters the black hole. Note that as $K < 2\Delta$, not all outgoing arcs can be used by at least two agents and there will be at least one v-vertex. □

5 Conclusion and Open Problems

We have shown that in the case of directed graphs, the cost (in the terms of the number of agents needed) of black hole search can be as high as exponential in the in-degree Δ of the black hole; furthermore this bound holds also in the case of synchronous agents. This is a rather striking difference from the undirected case where $\Delta + 1$ agents always suffice.

The main open problem here is whether the number of agents sufficient in the directed case is a function of only Δ and does not depend on the network size n. Our conjecture is that this is indeed the case, although the proof has eluded us so far.

Another research direction is to investigate how does the available structural information impact the number of agents needed for black hole location. We have made initial exploration, showing that giving a planar embedding of the planar graphs suffices to reduce the number of necessary agents to $2\Delta + 1$, and providing also a rather tight lower bound. Nevertheless, this area is mostly unexplored. For example, how much can Sense of Direction help here? What is the impact of having full topological knowledge? What is the weakest additional information sufficient to reduce the number of needed agents to a polynomial in Δ?

References

1. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. Kluwer, Dordrecht (2003)
2. Arkin, E., Bender, M., Fekete, S., Mitchell, J.: The freeze-tag problem: how to wake up a swarm of robots. In: 13^{th} ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), pp. 568–577 (2002)
3. Barriere, L., Flocchini, P., Fraigniaud, P., Santoro, N.: Capture of an intruder by mobile agents. In: Proc. 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA 2002), pp. 200–209 (2002)
4. Bender, M.A., Slonim, D.: The power of team exploration: Two robots can learn unlabeled directed graphs. In: 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), pp. 75–85 (1994)
5. Bender, M., Slonim, D.K.: The power of team exploration: two robots can learn unlabeled directed graphs. In: Proc. 35th Symp. on Foundations of Computer Science (FOCS 1994), pp. 75–85 (1994)
6. Bender, M.A., Fernández, A., Ron, D., Sahai, A., Vadhan, S.P.: The power of a pebble: Exploring and mapping directed graphs. In: STOC, pp. 269–278 (1998)
7. Budach, L.: On the solution of the labyrinth problem for finite automata. Elektronische Informationsverarbeitung und Kybernetic 11, 661–672 (1975)
8. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. ACM Transactions on Algorithms 4(4) (2008)
9. Czyzowicz, J., Kowalski, D., Markou, E., Pelc, A.: Searching for a black hole in tree networks. In: Higashino, T. (ed.) OPODIS 2004. LNCS, vol. 3544, pp. 67–80. Springer, Heidelberg (2005)
10. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. In: Proceedings: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, October 22–24, vol. 1, pp. 355–361 (1990); Formerly called the Annual Symposium on Switching and Automata Theory. IEEE catalog number 90CH29256. Computer Society order no. 2082
11. Dobrev, S., Flocchini, P., Kralovic, R., Prencipe, G., Ruzicka, P., Santoro, N.: Optimal search for a black hole in common interconnection networks. Networks 47(2), 61–71 (2006)
12. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Finding a black hole in an arbitrary network: optimal mobile agents protocols. In: Proc. of 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pp. 153–162 (2002)

13. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Finding a black hole in an arbitrary network: optimal mobile agents protocols. In: Proc. of 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pp. 153–162 (2002)
14. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Multiple agents rendezvous in a ring in spite of a black hole. In: Papatriantafilou, M., Hunel, P. (eds.) OPODIS 2003. LNCS, vol. 3144, pp. 34–46. Springer, Heidelberg (2004)
15. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. Algorithmica 48(1), 67–90 (2007)
16. Dobrev, S., Flocchini, P., Santoro, N.: Improved bounds for optimal black hole search in a network with a map. In: Kralovic, R., Sýkora, O. (eds.) SIROCCO 2004. LNCS, vol. 3104, pp. 111–122. Springer, Heidelberg (2004)
17. Esparza, O., Soriano, M., Munoz, J., Forne, J.: Host revocation authority: A way of protecting mobile agents from malicious hosts. In: Cueva Lovelle, J.M., Rodríguez, B.M.G., Gayo, J.E.L., Ruiz, M.d.P.P., Aguilar, L.J. (eds.) ICWE 2003. LNCS, vol. 2722. Springer, Heidelberg (2003)
18. Fraigniaud, P., Gasieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks 48(3), 166–177 (2006)
19. Fraigniaud, P., Ilcinkas, D.: Digraph exploration with little memory. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 246–257. Springer, Heidelberg (2004)
20. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. 345(2-3), 331–344 (2005)
21. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Impact of memory size on graph exploration capability. Discrete Applied Mathematics 156(12), 2310–2319 (2008)
22. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Tree exploration with advice. Inf. Comput. 206(11), 1276–1287 (2008)
23. Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: Bansal, N., Pruhs, K., Stein, C. (eds.) SODA, pp. 585–594. SIAM, Philadelphia (2007)
24. Greenberg, M., Byington, J., Harper, D.G.: Mobile agents and security. IEEE Commun. Mag. 36(7), 76–85 (1998)
25. Hoffmann, F.: One pebble does not suffice to search plane labyrinths. In: Gecseg, F. (ed.) FCT 1981. LNCS, vol. 117, pp. 433–444. Springer, Heidelberg (1981)
26. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence 117(2), 277–296 (2000)
27. Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Hardness and approximation results for black hole search in arbitrary graphs. In: Pelc, A., Raynal, M. (eds.) SIROCCO 2005. LNCS, vol. 3499, pp. 200–215. Springer, Heidelberg (2005)
28. Marco, G.D., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 271–282. Springer, Heidelberg (2005)
29. Oppliger, R.: Security issues related to mobile code and agent-based systems. Computer Communications 22(12), 1165–1170 (1999)
30. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. J. Algorithms 33, 281–295 (1999)
31. Sander, T., Tschudin, C.F.: Protecting mobile agents against malicious hosts. In: Vigna, G. (ed.) Mobile Agents and Security. LNCS, vol. 1419, pp. 44–60. Springer, Heidelberg (1998)

Optimal Probabilistic Ring Exploration by Semi-synchronous Oblivious Robots*

Stéphane Devismes[1], Franck Petit[2], and Sébastien Tixeuil[3]

[1] VERIMAG UMR 5104, Université Joseph Fourier, Grenoble (France)
stephane.devismes@imag.fr
[2] INRIA, LIP UMR 5668, Université de Lyon / ENS Lyon (France)
franck.petit@ens-lyon.fr
[3] Université Pierre et Marie Curie - Paris 6, INRIA Grand Large, France
sebastien.tixeuil@lip6.fr

Abstract. We consider a team of k identical, oblivious, semi-synchronous mobile robots that are able to sense (*i.e.*, view) their environment, yet are unable to communicate, and evolve on a constrained path. Previous results in this weak scenario show that initial symmetry yields high lower bounds when problems are to be solved by *deterministic* robots.

In this paper, we initiate research on probabilistic bounds and solutions in this context, and focus on the *exploration* problem of anonymous unoriented rings of any size. It is known that $\Theta(\log n)$ robots are necessary and sufficient to solve the problem with k deterministic robots, provided that k and n are coprime. By contrast, we show that *four* identical probabilistic robots are necessary and sufficient to solve the same problem, also removing the coprime constraint. Our positive results are constructive.

1 Introduction

We consider autonomous robots that are endowed with visibility sensors (but that are otherwise unable to communicate) and motion actuators. Those robots must collaborate to solve a collective task, namely *exploration*, despite being limited with respect to input from the environment, asymmetry, memory, etc. In this context, the exploration tasks requires every possible location to be visited by at least one robot, with the additional constraint that all robots stop moving after task completion.

Robots operate in *cycles* that comprise *look*, *compute*, and *move* phases. The look phase consists in taking a snapshot of the other robots positions using its visibility sensors. In the compute phase a robot computes a target destination based on the previous observation. The move phase simply consists in moving toward the computed destination using motion actuators.

The robots that we consider here have weak capacities: they are *anonymous* (they execute the same protocol and have no mean to distinguish themselves

* This work has been supported in part by the ANR projets *R-Discover* (08-ANR-CONTINT) and *SHAMAN*.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 195–208, 2010.

from the others), *oblivious* (they have no memory that is persistent between two cycles), and have no compass whatsoever (they are unable to agree on a common direction or orientation).

Related works. The vast majority of literature on coordinated distributed robots considers that those robots are evolving in a *continuous* two-dimentional Euclidian space and use visual sensors with perfect accuracy that permit to locate other robots with infinite precision, *e.g.*, [1,2,3,4,5,6].

Several works investigate restricting the capabilities of both visibility sensors and motion actuators of the robots, in order to circumvent the many impossibility results that appear in the general continuous model. In [7,8], robots visibility sensors are supposed to be accurate within a constant range, and sense nothing beyond this range. In [8,9], the space allowed for the motion actuator was reduced to a one-dimentional continuous one: a ring in [8], an infinite path in [9].

A recent trend was to shift from the classical continuous model to the *discrete* model. In the discrete model, space is partitioned into a *finite* number of locations. This setting is conveniently represented by a graph, where nodes represent locations that can be sensed, and where edges represent the possibility for a robot to move from one location to the other. Thus, the discrete model restricts both sensing and actuating capabilities of every robot. For each location, a robot is able to sense if the location is empty or if robots are positioned on it (instead of sensing the exact position of a robot). Also, a robot is not able to move from a position to another unless there is explicit indication to do so (*i.e.*, the two locations are connected by an edge in the representing graph). The discrete model permits to simplify many robot protocols by reasoning on finite structures (*i.e.*, graphs) rather than on infinite ones. However, as noted in most related papers [10,11,12,13], this simplicity comes with the cost of extra symmetry possibilities, especially when the authorized paths are also symmetric (indeed, techniques to break formation such as those of [5] cannot be used in the discrete model).

Assuming visibility capabilities, the two main problems that have been studied in the discrete robot model are gathering [10,11] and exploration [12,13]. For gathering, both breaking symmetry [10] and preserving symmetry are meaningful approaches. For exploration, the fact that robots need to stop after exploring all locations requires robots to "remember" how much of the graph was explored, *i.e.*, be able to distinguish between various stages of the exploration process since robots have no persistent memory. As configurations can be distinguished only by robot positions, the main complexity measure is then the number of robots that are needed to explore a given graph. The vast number of symmetric situations induces a large number of required robots. For tree networks, [13] shows that $\Omega(n)$ robots are necessary for most n-sized tree, and that sublinear robot complexity (actually $\Theta(\log n / \log\log n)$) is possible only if the maximum degree of the tree is 3. In uniform rings, [12] proves that the necessary and sufficient number of robots is $\Theta(\log n)$, although it proposes an algorithm that works with an additional assumption: the number k of robots and the size n of the ring are coprime. Note that all previous approaches in the discrete model are

deterministic, *i.e.*, if a robot is presented twice the same situation, its behavior is the same in both cases.

Our contribution. In this paper, we consider the *semi-synchronous model* introduced in [14]. It is straighforward to see that the necessary conditions and bounds exposed in [12] for the deterministic exploration still hold in the semi-synchronous model. Here we propose to adopt a *probabilistic* approach to lift constraints and to obtain tigher bounds. By constrast with the deterministic approach, we show that *four* identical probabilistic robots are necessary and sufficient to solve the exploration problem in any anonymous unoriented ring of size $n > 8$, also removing the coprime constraint between the number of robots and the size of the ring. Our negative result show that for any ring of size at least four, there cannot exist any protocol with three robots in our setting, even if they are allowed to make use of probabilistic primitives. Our positive results are constructive, as we present a randomized protocol with four robots for any ring of size more than eight.

Outline. The remaining of the paper is divided as follows. Section 2 presents the system model that we use throughout the paper. Section 3 provides evidence that no three probabilistic robots can explore every ring, while Section 4 presents our protocol with four robots. Section 5 gives some concluding remarks.

For space consideration, several technical proofs are omitted, see the technical report for details ([15], `http://hal.inria.fr/inria-00360305/fr/`).

2 Model

Distributed System. We consider systems of autonomous mobile entities called *agents* or *robots* evolving into a *graph*. We assume that the graph is a *ring* of n nodes, $u_0, \ldots, u_{n-1}$, *i.e.*, u_i is connected to both u_{i-1} and u_{i+1} — every computation over indices is assumed to be modulus n. The indices are used for notation purposes only: the nodes are *anonymous* and the ring is *unoriented*, *i.e.*, given two neighboring nodes u, v, there is no kind of explicit or implicit labelling allowing to determine whether u is on the right or on the left of v. Operating in the ring are $k \leq n$ anonymous robots.

A *protocol* is a collection of k *programs*, one operating on each robot. The program of a robot consists in executing *Look-Compute-Move cycles* infinitely many times. That is, the robot first observes its environment (Look phase). Based on its observation, a robot then (probabilistically or deterministically) decides — according to its program — to move or stay idle (Compute phase). When a robot decides a move, it moves to its destination during the Move phase.

The robots do not communicate in an explicit way; however they see the position of the other robots and can acquire knowledge from this information. We assume that the robots cannot remember any previous observation nor computation performed in any previous step. Such robots are said to be *oblivious* (or *memoryless*). The robots are also *uniform* and *anonymous*, i.e, they all have

the same program using no local parameter (such that an identity) allowing to differentiate any of them.

Computations. We consider a *semi-synchronous* model similar to the one in [14]. In this model, time is represented by an infinite sequence of instants $0, 1, 2, \ldots$ At every instant $t \geq 0$, a non-empty subset of robots is activated to execute a cycle. The execution of each cycle is assumed to be *atomic*: Every robot that is activated at instant t instantaneously executes a full cycle between t and $t + 1$. Atomicity guarantees that at any instant the robots are on some nodes of the ring but not on edges. Hence, during a Look phase, a robot sees no robot on edges.

We assume that during the Look phase, every robot can perceive whether several robots are located on the same node or not. This ability is called *Multiplicity Detection*. We shall indicate by $d_i(t)$ the multiplicity of robots present in node u_i at instant t. More precisely $d_i(t) = j$ indicates that there are j robots in node u_i at instant t. If $d_i(t) \geq 2$, then we say that there is a *tower* in u_i at instant t (or simply there is a *tower* in u_i when it is clear from the context). We say a node u_i is *free at instant* t (or simply *free* when it is clear from the context) if $d_i(t) = 0$. Conversely, we say that u_i is *occupied at instant* t (or simply *occupied* when it is clear from the context) if $d_i(t) \neq 0$.

Given an arbitrary orientation of the ring and a node u_i, $\gamma^{+i}(t)$ (respectively, $\gamma^{-i}(t)$) denotes the sequence $\langle d_i(t)d_{i+1}(t)\ldots d_{i+n-1}(t)\rangle$ (resp., $\langle d_i(t)d_{i-1}(t)\ldots d_{i-(n-1)}(t)\rangle$). The sequence $\gamma^{-i}(t)$ is called *mirror* of $\gamma^{+i}(t)$ and conversely. Since the ring is unoriented, agreement on only one of the two sequences $\gamma^{+i}(t)$ and $\gamma^{-i}(t)$) is impossible. The (unordered) pair $\{\gamma^{+i}(t), \gamma^{-i}(t)\}$ is called the *view* of node u_i at instant t (we omit "at instant t" when it clear from the context). The view of u_i is said to be *symmetric* if and only if $\gamma^{+i}(t) = \gamma^{-i}(t)$. Otherwise, the view of u_i is said to be *asymmetric*.

By convention, we state that the *configuration* of the system at instant t is $\gamma^{+0}(t)$. Any configuration from which there is a probability 0 that a robot moves is said to be *terminal*. Let $\gamma = \langle x_0 x_1 \ldots x_{n-1}\rangle$ be a configuration. The configuration $\langle x_i x_{i+1} \ldots x_{i+n-1}\rangle$ is obtained by rotating γ of $i \in [0 \ldots n-1]$. Two configurations γ and γ' are said to be *indistinguishable* if and only if γ' can be obtained by rotating γ or its mirror. Two configurations that are not indistinguishable are said to be *distinguishable*. We designate by *initial configurations* the configurations from which the system can start at instant 0.

During the Look phase of some cycle, it may happen that both edges incident to a node v currently occupied by the robot look identical in the snapshot, *i.e.*, v lies on a symmetric axis of the configuration. In this case, if the robot decides to move, it may traverse any of the two edges. We assume the worst case decision in such cases, *i.e.*, that the decision to traverse one of these two edges is taken by an adversary.

We call *computation* any infinite sequence of configurations $\gamma_0, \ldots, \gamma_t, \gamma_{t+1}, \ldots$ such that (1) γ_0 is a possible initial configuration and (2) for every instant $t \geq 0$, γ_{t+1} is obtained from γ_t after some robots (at least one) execute a cycle. Any transition γ_t, γ_{t+1} is called a step of the computation. A computation c *terminates* if c contains a terminal configuration.

A *scheduler* is a predicate on computations, that is, a scheduler defines a set of *admissible* computations, such that every computation in this set satisfies the scheduler predicate. Here we assume a *distributed fair* scheduler. Distributed means that, at every instant, any non-empty subset of robots can be activated. Fair means that every robot is activated infinitively often during a computation. A particular case of distributed fair scheduler is the *sequential* fair scheduler: at every instant, one robot is activated and every robot is activated infinitively often during a computation. In the following, we call *sequential computation* any computation that satisfies the sequential fair scheduler predicate.

Problem to be solved. We consider the *exploration* problem, where k robots collectively explore a n-sized ring before stopping moving forever. More formally, a protocol $\mathcal{P}$ *deterministically* (resp. *probabilistically*) solves the exploration problem if and only if every computation c of $\mathcal{P}$ starting from a *towerless configuration* satisfies:

1. c terminates in *finite time* (resp. with *expected finite time*).
2. Every node is visited by at least one robot during c.

The previous definition implies that every initial configuration of the system in the problem we consider is *towerless*. Using probabilistic solutions, termination is not certain, however the overall probability of non-terminating computations is 0.

3 Negative Result

In this section, we show that the exploration problem is impossible to solve in our settings (*i.e.*, oblivious robots, anonymous ring, distributed scheduler, ...) if there is less than four robots, even in a probabilistic manner (Corollary 2). The proof is made in two steps:

- The first step is based on the fact that obliviousness constraints any exploration protocol to construct an implicit memory using the configurations. We show that if the scheduler behaves sequentially, then in any case except one, it is not possible to particularize enough configurations to memorize which nodes have been visited (Theorem 1 and Lemma 4).
- The second step consists in excluding the last case (Theorem 2).

If $n > k$, any terminal configuration should be distinguishable from any possible initial (towerless) configuration. Hence, follows:

Remark 1. If $n > k$, any terminal configuration of any exploration protocol contains at least one tower.

Lemmas 1 to 3 proven below are technical results that lead to Corollary 1. The latter exhibits the minimal size of a subset of particular configurations required to solve the exploration problem.

Definition 1 (MRP). *Let s be a sequence of configurations. The* minimal relevant prefix *of s, noted $\mathcal{MRP}(s)$, is the maximal subsequence of s where no two consecutive configurations are identical.*

Lemma 1. *Let $\mathcal{P}$ be any (probabilistic or deterministic) exploration protocol for k robots in a ring of $n > k$ nodes. For every sequential computation c of $\mathcal{P}$ that terminates, $\mathcal{MRP}(c)$ has at least $n - k + 1$ configurations containing a tower.*

Proof. Assume, by the contradiction, that there is a sequential computation c of $\mathcal{P}$ that terminates and such that $\mathcal{MRP}(c)$ has less than $n-k+1$ configurations containing a tower.

Take the last configuration α without tower which appear in computation c and all remaining configurations (all of them contains towers) that follow in c and form c'. As α could be an initial configuration and c is an admissible sequential computation that terminates, c' is also an admissible sequential computation of $\mathcal{P}$ that terminates. Notice that $\mathcal{MRP}(c')$ has at most $n - k + 1$ configurations. Since c' is sequential, going from configuration α to a configuration with towers, no new nodes are explored (the same happens when remaining at the same configuration with towers). Hence the total number of nodes explored upon the termination of c' is at most k (the ones that are initially visited) $+ n - k - 1$ (the ones that are dynamically visited) $= n - 1$: c' terminates before all nodes are visited, a contradiction.

Lemma 2. *Let $\mathcal{P}$ be any (probabilistic or deterministic) exploration protocol for k robots in a ring of $n > k$ nodes. For every sequential computation c of $\mathcal{P}$ that terminates, $\mathcal{MRP}(c)$ has at least $n - k + 1$ configurations containing a tower of less than k robots.*

Proof. Assume, by the contradiction, that there is a sequential computation c of $\mathcal{P}$ that terminates and such that $\mathcal{MRP}(c)$ has less than $n-k+1$ configurations containing a tower of less than k robots.

Take the last configuration α without tower which appear in computation c and all remaining configurations (all of them contains towers) that follow in c and form c'. As α could be an initial configuration and c is an admissible sequential computation that terminates, c' is also an admissible sequential computation of $\mathcal{P}$ that terminates.

$\mathcal{MRP}(c')$ is constituted of a configuration with no tower followed by at least $n - k + 1$ configurations containing a tower by Lemma 1 and $n - k$ new nodes (remember that k nodes are already visited in the initial configuration) must be visited before c' reaches its terminal configuration.

Consider a step $\alpha\alpha'$ in c'.

- If $\alpha = \alpha'$, then no node is visited during the step.
- If $\alpha \neq \alpha'$, then there are three possible cases:
  1. *α contains no towers.* In this case, α is the initial configuration and α' contains a tower. As only one robot moves in $\alpha\alpha'$ to create a tower (c' is sequential), no node is visited during this step.

2. *α contains a tower and α' contains a tower of k robots.* As c' is sequential and all robots are located at the same node in α', one robot moves to an already occupied node in $\alpha\alpha'$ and no node is visited during this step.
3. *α contains a tower and α' contains a tower of less than k robots.* In this case, at most one node is visited in $\alpha\alpha'$ because c' is sequential.

To sum up, only the steps from a configuration containing a tower to a configuration containing a tower of less than k robots allow to visit at most one node each time. Now, in $\mathcal{MRP}(c')$ there are less than $n-k+1$ configurations containing a tower of less than k robots and the first of these configurations appearing into c' is consecutive to a step starting from the initial configuration. Hence, less than $n-k$ nodes are dynamically visited during c' and, as exactly k nodes are visited in the initial configuration, less than n nodes are visited when c' terminates, a contradiction.

Lemma 3. *Let $\mathcal{P}$ be any (probabilistic or deterministic) exploration protocol for k robots in a ring of $n > k$ nodes. For every sequential computation c of $\mathcal{P}$ that terminates, $\mathcal{MRP}(c)$ has at least $n-k+1$ configurations containing a tower of less than k robots and any two of them are distinguishable.*

Proof. Consider any sequential computation c of $\mathcal{P}$ that terminates.

By Lemma 2, $\mathcal{MRP}(c)$ has x configurations containing a tower of less than k robots where $x \geq n-k+1$.

We first show that (**) *if c contains at least two different configurations having a tower of less than k robots that are indistinguishable, then there exists a sequential computation c' that terminates and such that $\mathcal{MRP}(c')$ has x' configurations containing a tower of less than k robots where $x' < x$.* Assume that there are two different indistinguishable configurations γ and γ' in c having a tower of less than k robots. Without loss of generality, assume that γ occurs at time t in c and γ' occurs at time $t' > t$ in c. Consider the two following cases:

1. **γ' can be obtained by applying a rotation of i to γ.** Let p be the prefix of c from instant 0 to instant t. Let s be the suffix of c starting at instant $t'+1$. Let s' be the sequence obtained by applying a rotation of $-i$ to the configurations of s. As the ring and the robots are anonymous, ps' is an admissible sequential computation that terminates. Moreover, by construction $\mathcal{MRP}(ps')$ has x' configurations containing a tower of less than k robots where $x' < x$. Hence (**) is verified in this case.
2. **γ' can be obtained by applying a rotation of i to the mirror of γ.** We can prove (**) in this case by slightly modifying the proof of the previous case: we have just to apply the rotation of $-i$ to the *mirrors* of the configurations of s.

By (**), if $\mathcal{MRP}(c)$ contains less than $n-k+1$ distinguishable configurations with a tower of less than k robots, it is possible to (recursively) construct an admissible computation c' of $\mathcal{P}$ that terminates such that $\mathcal{MRP}(c')$ has less than $n-k+1$ configurations containing a tower of less than k robots, a contradiction to Lemma 2. Hence, the lemma holds.

From Lemma 3, we can deduce the following corollary:

Corollary 1. *Considering any (probabilistic or deterministic) exploration protocol for k robots in a ring of $n > k$ nodes, there exists a subset $\mathcal{S}$ of at least $n - k + 1$ configurations such that:*

1. *Any two different configurations in $\mathcal{S}$ are distinguishable, and*
2. *In every configuration in $\mathcal{S}$, there is a tower of less than k robots.*

Theorem 1. *$\forall k, 0 \leq k < 3, \forall n > k$, there is no exploration protocol (even probabilistic) of a n-size ring with k robots.*

Proof. First, for $k = 0$, the theorem is trivially verified. Consider then the case $k = 1$ and $k = 2$: with one robot it is impossible to construct a configuration with one tower; with two robots it is impossible to construct a configuration with one tower of less than k robots ($k = 2$). Hence, for $k = 1$ and $k = 2$, the theorem is a direct consequence of Corollary 1.

Lemma 4. *$\forall n > 4$, there is no exploration protocol (even probabilistic) of a n-size ring with three robots.*

Proof. With three robots, the size of the maximal set of distinguishable configurations containing a tower of less than three robots is $\lfloor n/2 \rfloor$. By Corollary 1, we have then the following inequality:

$$\lfloor n/2 \rfloor \geq n - k + 1$$

From this inequality, we can deduce that n must be less of equal than four and we are done.

From this point on, we know that, assuming $k < 4$, Corollary 1 prevents the existence of any exploration protocol in any case except one: $k = 3$ and $n = 4$ (Theorem 1 and Lemma 4). Actually, assuming that the scheduler is sequential is not sufficient to show the impossibility in this latter case: Indeed, there is an exploration protocol for $k = 3$ and $n = 4$ if we assume a sequential scheduler. This latter protocol can be found in the technical report ([15], `http://hal.inria.fr/inria-00360305/fr/`).

The theorem below is obtained by showing the impossibility for $k = 3$ and $n = 4$ using a (non-sequential) distributed scheduler.

Theorem 2. *There is no exploration protocol (even probabilistic) of a n-size ring with three robots for every $n > 3$.*

Proof Outline. Lemma 4 excludes the existence of any exploration protocol for three robots in a ring of $n > 4$ nodes. Hence, to show this theorem, we just have to show that there is no exploration protocol for three robots working in a ring of four nodes.

The remainder of the proof consists in a combinatorial study of all possible protocols for $k = 3$ robots and $n = 4$ nodes. In each case, we show that the protocol leads to one of the following contradiction:

– Either, the adversarial choices of the scheduler allow to construct an admissible computation that never terminates with probability 1.
– Or, for every possible terminal configuration (*i.e.*, any configuration containing a tower, see Remark 1), there is an admissible computation that reaches the terminal configuration without visiting all nodes. □

From Theorems 1 and 2, we can deduce the following corollary:

Corollary 2. $\forall k, 0 \leq k < 4, \forall n > k$, *there is no exploration protocol (even probabilistic) of a n-size ring with k robots.*

4 Positive Result

In this section, we propose a probabilistic exploration protocol for $\boldsymbol{k = 4}$ robots in a ring of $\boldsymbol{n > 8}$ nodes. We first define some useful terms in Subsection 4.1. We then give the general principle of the protocol in Subsection 4.2. Finally, we detail and prove the protocol in Subsection 4.3.

4.1 Definitions

Below, we define some terms that characterize the configurations.

We call *segment* any maximal non-empty elementary path of occupied nodes. The *length of a segment* is the number of nodes that compose it. We call *x-segment* any segment of length x. In the segment $s = u_i, \ldots, u_k$ $(k \geq i)$ the nodes u_i and u_k are termed as the *extremities* of s. An *isolated node* is a node belonging to a 1-segment.

We call *hole* any maximal non-empty elementary path of free nodes. The *length of a hole* is the number of nodes that compose it. We call *x-hole* any hole of length x. In the hole $h = u_i, \ldots, u_k$ $(k \geq i)$ the nodes u_i and u_k are termed as the *extremities* of h. We call *neighbor* of an hole any node that does not belong to the hole but is neighbor of one of its extremities. In this case, we also say that the hole is a *neighboring hole* of the node. By extension, any robot that is located at a neighboring node of a hole is also referred to as a neighbor of the hole.

We call *arrow* a maximal elementary path $u_i, \ldots, u_k$ of length at least four such that (i) u_i and u_k are occupied by one robot, (ii) $\forall j \in [i+1 \ldots k-2]$, u_j is free, and (iii) there is a tower of two robots in u_{k-1}. The node u_i is called the *arrow tail* and the node u_k is called the *arrow head*. The *size* of an arrow is the number of free nodes that compose it, *i.e.*, it is the length of the arrow path minus 3. Note that the minimal size of an arrow is 1 and the maximal size is $n-3$. Note also that when there is an arrow in a configuration, the arrow is unique. An arrow is said to be *primary* if its size is 1. An arrow is said to be *final* if its size is $n-3$.

Figure 1 illustrates the notion of arrows: In Configuration (i) the arrow is formed by the path u_4, u_5, u_0, u_1; the arrow is primary; the node u_4 is the tail

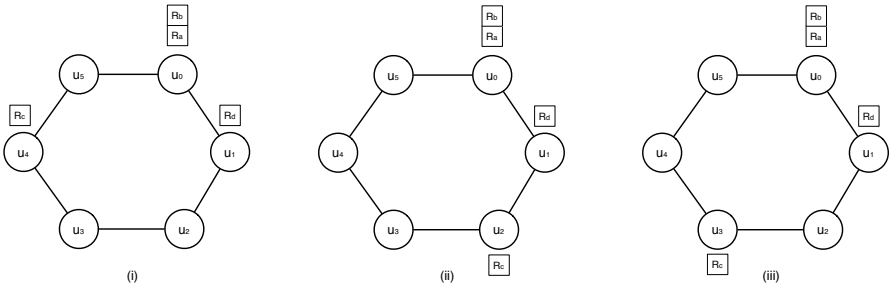


Fig. 1. Arrows

and the node u_1 is the head. In Configuration (ii), there is a final arrow (the path u_2, u_3, u_4, u_5, u_0, u_1). Finally, the size of the arrow in Configuration (iii) (the path u_3, u_4, u_5, u_0, u_1) is 2.

4.2 Overview of the Solution

Our protocol (Algorithm 1) proceeds in three distinct phases:

- **Phase I:** Starting from a configuration without tower, the robots move along the ring in such a way that (i) they never form any tower and (2) form a unique segment (a 4-segment) in finite expected time.
- **Phase II:** Starting from a configuration with a unique segment, the four robots form a primary arrow in finite expected time. The 4-segment is maintained until the primary arrow is formed.
- **Phase III:** Starting from a configuration where the four robots form a primary arrow, the arrow tail deterministically moves toward the arrow head in such way that the length of the arrow never decreases. The protocol terminates when robots form a final arrow. At the termination, all nodes have been visited.

Note that the protocol we propose is probabilistic. As a matter of fact, as long as possible the robots move deterministically. However, we use randomization to break the symmetry in some cases: When the system is in a symmetric configuration, the scheduler may choose to synchronously activate some processes in such way that the system stays in a symmetric configuration. To break the symmetry despite the choice of the scheduler, we proceed as follows: The activated robots toss a coin (with a uniform probability) during their Compute phase. If they win the toss, they decide to move, otherwise they decide to stay idle. In this case, we say that the robots **try to move**. Conversely, when a process deterministically decides to move in its Compute phase, we simply say that the process **moves**.

Algorithm 1. The protocol.

```
1: if the four robots do not form a final arrow then
2:     if the configuration contains neither an arrow nor a 4-segment then
3:         Execute Procedure Phase I;
4:     else
5:         if the configuration contains a 4-segment then
6:             Execute Procedure Phase II;
7:         else      /* the configuration contains an arrow */
8:             Execute Procedure Phase III;
```

4.3 Detailed Description of the Solution

Phase I. Phase I is described in Algorithm 2. The aim of this phase is to eventually form a 4-segment without creating any tower during the process. Roughly speaking, in asymmetric configurations, robots moves determiniscally (Lines 4, 10, 27, 31). By contrast, in symmetric configurations, robots moves probabilistically using **Try to move** (Lines 16 and 22). Note that in all cases, we prevent the tower formation by applying the following constraint: a robot can move through a neighboring hole $\mathcal{H}$ only if its length is at least 2 or if the other neighboring robot cannot move through $\mathcal{H}$. Hence, we obtain the following lemma:

Lemma 5. *If the configuration at instant t contains neither a 4-segment nor a tower, then the configuration at instant $t+1$ contains no tower.*

The probabilistic convergence to a 4-segment is guaranteed by the fact that in a symmetric configuration, the moving robots move probabilistically. Thanks to that, the symmetries are eventually broken and the system reaches an asymmetric configuration from which the robots deterministically move until forming a 4-segment. Hence, we obtain the lemma below:

Lemma 6. *Starting from any initial (towerless) configuration, the system reaches in finite expected time a configuration containing a 4-segment.*

Phase II. Phase II is described in Algorithm 3. Starting from a configuration where there is a 4-segment on nodes u_i, u_{i+1}, u_{i+2}, u_{i+3}, the system eventually reaches a configuration where a primary arrow is formed on nodes u_i, u_{i+1}, u_{i+2}, u_{i+3}. To that goal, we proceed as follows: Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the robots located at the nodes u_{i+1} and u_{i+2} of the 4-segment. $\mathcal{R}_1$ and $\mathcal{R}_2$ try to move to u_{i+2} and u_{i+1}, respectively. Eventually only one of these robots moves and we are done. Hence, we have the two lemmas below:

Lemma 7. *Let γ be a configuration containing a 4-segment $u_i, u_{i+1}, u_{i+2}, u_{i+3}$. If γ is the configuration at instant t, then the configuration at instant $t+1$ is either identical to γ or the configuration containing the primary arrow u_i, u_{i+1}, u_{i+2}, u_{i+3}.*

Lemma 8. *From a configuration containing a 4-segment, the system reaches a configuration containing a primary arrow in finite expected time.*

Algorithm 2. Procedure *Phase* I.

```
1: if the configuration contains a 3-segment then
2:     begin
3:     if I am the isolated robot then
4:         Move toward the 3-segment through the shortest hole;
5:     end
6: else
7:     if the configuration contains a unique 2-segment then      /* Two robots are isolated */
8:         begin
9:         if I am at the closest distance from the 2-segment then
10:            Move toward the 2-segment through the hole having me and an extremity of the
    2-segment as neighbors;
11:        end
12:    else
13:        if the configuration contains (exactly) two 2-segments then
14:            begin
15:            if I am a neighbor of a longuest hole then
16:                Try to move toward the other 2-segment through my neighboring hole;
17:            end
18:        else     /* the four robots are isolated */
19:            begin
20:            Let l_max be the length of the longuest hole;
21:            if every robot is neighbor of a l_max-hole then
22:                Try to move through a neighboring l_max-hole;
23:            else
24:                if 3 robots are neighbors of a l_max-hole then
25:                    begin
26:                    if I am neighbor of only one l_max-hole then
27:                        Move toward the robot that is neighbor of no l_max-hole through my short-
    est neighboring hole;
28:                    end
29:                else     /* 2 robots are neighbors of the unique l_max-hole */
30:                    if I am neighbor of the unique l_max-hole then
31:                        Move through my shortest neighboring hole;
32:            end
```

Algorithm 3. Procedure *Phase* II.

```
1: if I am not located at an extremity of the 4-segment then
2:     Try to move toward my neighboring node that is not an extremity of the 4-segment;
```

Algorithm 4. Procedure *Phase* III.

```
1: if I am the arrow tail then
2:     Move toward the arrow head through the hole having me and the arrow head as neighbor;
```

Phase III. Phase III is described in Algorithm 4. This phase is fully deterministic: This phase begins when there is a primary arrow. Let $\mathcal{H}$ be the hole between the tail and the head of arrow at the beginning of the phase. From the previous phase, we know that all nodes forming the primary arrow are already visited. So, the unvisited nodes can only be on $\mathcal{H}$ and the phase just consists in traversing $\mathcal{H}$. To that goal, the robot located at the arrow tail traverses $\mathcal{H}$. When it is done, the system is in a terminal configuration containing a final arrow and all nodes have been visited. Hence, we can conclude with the following theorem:

Theorem 3. *Algorithm 1 is a probabilistic exploration protocol for* $\mathbf{4}$ *robots in a ring of* $\boldsymbol{n > 8}$ *nodes.*

5 Conclusion

We considered a semi-synchronous model of computation. In this model, we provided evidence that for the exploration problem in uniform rings, randomization could shift complexity from $\Theta(\log n)$ to $\Theta(1)$. While applying randomization to other problem instances is an interesting topic for further research, we would like to point out immediate open questions raised by our work:

1. Though we were able to provide a general algorithm for any n (strictly) greater than eight, it seems that ad hoc solutions have to be designed when n is between five and eight (inclusive).
2. Our protocol is optimal with respect to the number of robots. However, the efficiency (in terms of exploring time) is only proved to be finite. Actually computing the convergence time from our proof argument is feasible, but it would be more interesting to study how the number of robots relates to the time complexity of exploration, as it seems natural that more robots will explore the ring faster.
3. It is worth investigating if our results can be extended to the (full) asynchronous model.

Acknowledgments. We are grateful to the anonymous referees for a very careful reading of the manuscript and a number of valuable remarks and suggestions that enabled us to improve the quality of the paper.

References

1. Asahiro, Y., Fujita, S., Suzuki, I., Yamashita, M.: A self-stabilizing marching algorithm for a group of oblivious robots. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 125–144. Springer, Heidelberg (2008)
2. Souissi, S., Défago, X., Yamashita, M.: Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. TAAS 4(1) (2009)
3. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM J. Comput. 28(4), 1347–1363 (1999)
4. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. Theor. Comput. Sci. 407(1-3), 412–447 (2008)
5. Dieudonné, Y., Labbani-Igbida, O., Petit, F.: Circle formation of weak mobile robots. TAAS 3(4) (2008)
6. Dieudonné, Y., Petit, F.: Scatter of weak mobile robots. Parallel Processing Letters 19(1), 175–184 (2009)
7. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Transactions on Robotics and Automation (1999)
8. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theor. Comput. Sci. 337(1-3), 147–168 (2005)
9. Bouzid, Z., Potop-Butucaru, M.G., Tixeuil, S.: Byzantine-resilient convergence in oblivious robot networks. In: Garg, V., Wattenhofer, R., Kothapalli, K. (eds.) ICDCN 2009. LNCS, vol. 5408, Springer, Heidelberg (2008)

10. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theor. Comput. Sci. 390(1), 27–39 (2008)
11. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: Gathering of asynchronous oblivious robots on a ring. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 446–462. Springer, Heidelberg (2008)
12. Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Computing without communicating: Ring exploration by asynchronous oblivious robots. In: Tovar, E., Tsigas, P., Fouchal, H. (eds.) OPODIS 2007. LNCS, vol. 4878, pp. 105–118. Springer, Heidelberg (2007)
13. Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Remembering without memory: Tree exploration by asynchronous oblivious robots. In: Shvartsman, A.A., Felber, P. (eds.) SIROCCO 2008. LNCS, vol. 5058, pp. 33–47. Springer, Heidelberg (2008)
14. Efrima, A., Peleg, D.: Distributed algorithms for partitioning a swarm of autonomous mobile robots. Theor. Comput. Sci. 410(14), 1355–1368 (2009)
15. Devismes, S., Petit, F., Tixeuil, S.: Optimal probabilistic ring exploration by asynchronous oblivious robots. Technical Report inria-00360305, INRIA (February 2009)

Revisiting Randomized Parallel Load Balancing Algorithms

Guy Even and Moti Medina

School of Electrical Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel
{guy,medinamo}@eng.tau.ac.il

Abstract. We deal with the well studied allocation problem of assigning n balls to n bins so that the maximum number of balls assigned to the same bin is minimized. We focus on randomized, constant-round, distributed, asynchronous algorithms for this problem.

Adler et al. [1] presented lower bounds and upper bounds for this problem. A similar lower bound appears in Berenbrink et al. [2]. The lower bound is based on a topological assumption. Our first contribution is the observation that the topological assumption does not hold for two algorithms presented by Adler et al. [1]. We amend this situation by presenting direct proofs of the lower bound for these two algorithms.

We present an algorithm in which a ball that was not allocated in the first round retries with a new choice in the second round. We present tight bounds on the maximum load obtained by our algorithm. The analysis is based on analyzing the expectation and transforming it to a bound with high probability using martingale tail inequalities.

Finally, we present a 3-round heuristic with a single synchronization point. We conducted experiments that demonstrate its advantage over parallel algorithms for $10^6 \leq n \leq 10^8$ balls and bins. In fact, the obtained maximum load meets the best results for sequential algorithms.

Keywords: static randomized parallel allocation, load balancing, balls and bins, martingales.

1 Introduction

Azar et al. [3] considered the problem of allocating balls to bins in a balanced way. For simplicity, suppose that the number of balls equals the number of bins, and is denoted by n. If each ball selects a bin uniformly and independently at random, then with high probability (w.h.p.)[1] the maximum load of a bin is $\Theta(\ln n / \ln \ln n)$ (see Thm. 2). Azar et al. proved that, if balls choose two random bins and each ball is sequentially placed in a bin that is less loaded among the two, then w.h.p. the maximum load is only $\ln \ln n / \ln 2 + \Theta(1)$.

This surprising improvement in the maximum load has spurred a lot of interest in randomized load balancing in various settings. Adler et al. [1] studied parallel, distributed, asynchronous, load-balancing algorithms. They presented bounds of $\Theta(\sqrt[r]{\ln n / \ln \ln n})$ for parallel load balancing using r rounds of communication.

[1] We say that an event X occurs *with high probability* if $\Pr(X) \geq 1 - O\left(\frac{1}{n}\right)$.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 209–221, 2010.

Another parallel algorithm with the same asymptotic bounds was presented by Stemann [4] with a single synchronization point. Berenbrink et al. [2] generalized to $r \leq \log \log n$ communication rounds and to weighted balls.

1.1 The Model for Parallel Randomized Load Balancing Algorithms

We overview the model of parallel load balancing from Adler et al. [1]. There are n balls and n bins. In the beginning, each ball chooses d (a constant number) bins independently and uniformly at random (i.u.r).

The communication graph is a bipartite graph between balls and bins. Each ball is connected by edges to the d bins it has chosen. Messages are sent only along edges in the communication graph. Communication proceeds in *rounds*. Each round consists of messages from balls to bins and responses from bins to balls. We assume that each node (i.e., ball or bin) may simultaneously send messages to all its neighbors. In the last round, each ball commits to one of the d bins that it has chosen initially.

Adler et al. were interested in asynchronous algorithms. This means that a node may wait for a message only if the message is guaranteed to be sent to it. In particular, arrival of messages may be delayed so that messages from later rounds may precede messages from earlier rounds.

Finally, the model requires symmetry which we formalize as follows. For every execution σ of the algorithm, and for any permutation π of the balls and bins (i.e. renaming), the corresponding execution $\pi(\sigma)$ is a valid execution of the algorithm.

1.2 Previous Algorithms

The greedy algorithm. The greedy algorithm for load balancing presented in [3] is a sequential algorithm. Each ball, in its turn, chooses d bins uniformly and independently at random. The ball queries each of these bins for its current load (i.e., the number of balls that are assigned to it). The ball is placed in a bin with the minimum load. Azar et al. [3] proved that w.h.p. the maximum load at the end of this process is $\ln \ln n / \ln d + \Theta(1)$.

The parallel greedy algorithm: PGREEDY**.** Adler et al. [1] presented and investigated algorithm PGREEDY described below. Adler et. al [1] proved that w.h.p. the maximum load achieved by PGREEDY is $O(\sqrt{\ln n / \ln \ln n})$. They also proved a matching lower bound. For simplicity, we present the version in which each ball chooses $d = 2$ bins. We denote the balls by $b \in [1..n]$ and the bins by $u \in [1..n]$. The algorithm works as follows:

1. Each ball b chooses two bins $u_1(b)$ and $u_2(b)$ independently and uniformly at random. The ball b sends requests to bins $u_1(b)$ and $u_2(b)$.
2. Upon receiving a request from ball b, bin u responds to ball b by reporting the number of requests it has received so far. We denote this number by $h_u(b)$, and refer to it as the *height* of ball b in bin u.

3. After receiving its heights from $u_1(b)$ and $u_2(b)$, ball b sends a commit to the bin that assigned a lower height. (Tie-breaking rules are not addressed in [1].)

The threshold algorithm: THRESHOLD**.** The algorithm THRESHOLD studied by Adler et al. [1] works differently. Two parameters define the algorithm: a threshold parameter T bounds the number of balls that may be assigned to each bin in each round, and r bounds the number of rounds. Initially, all balls are unaccepted. In each round, each unaccepted ball chooses independently and uniformly a single random bin. Each bin accepts the first T balls that have chosen it. The other balls, if any, receive a rejection.

Note that, although described "in rounds", algorithm THRESHOLD can work completely asynchronously as distinct rounds may run simultaneously. Adler et al. prove that, the number of unaccepted balls decreases rapidly, and thus, if r is constant, then setting $T = O(\sqrt[r]{\ln n / \ln \ln n})$ requires w.h.p. at most r rounds. They also proved a maximum load of $\Theta(r)$ for $T = 1$ and $r = \log \log n$ rounds.

1.3 Lower Bounds

In [1,2] a lower bound of $\Omega(\sqrt[r]{\ln n / \ln \ln n})$ was proved for the maximum load obtained by parallel randomized load balancing algorithms, where r denotes the number of rounds and n denotes the number of bins and balls. This lower bound holds for a constant d and $r \leq \log \log n$.

The lower bound uses a random (hyper)graph, called the *access graph*, that represents the choices of the balls. A structure called a *witness tree* is proved to exist in the access graph with constant probability [1,5].

The lower bound is based on a topological assumption (see Assumption 1) that the final commitment of a ball is based only on the topology of the neighborhood of radius $(r - 1)$ in the access graph (see Sect. 2 for details).

1.4 Contributions

Gaps in applying the lower bound. Although in [1] it is stated that the topological assumption holds for algorithm THRESHOLD, we show that the topological assumption does not hold for algorithms PGREEDY and THRESHOLD. The reason the assumption does not hold is that the commitment is based on information not included in the topology of the access graph (e.g., heights and round numbers). Since the lower bound in [1,2] is based on the topological assumption, and since it natural to design algorithms that violate this assumption, the question of proving general lower bounds for the maximum load in parallel randomized load balancing is reopened.

Lower bounding PGREEDY **and** THRESHOLD**.** We show how the witness tree technique can be used to prove the $\Omega(\sqrt[r]{\frac{\ln n}{\ln \ln n}})$ lower bound for the PGREEDY and THRESHOLD algorithms. These proofs are not based on the topological assumption as in [1,2]. Instead, it is proved that high load is obtained with at least

constant probability conditioned on the existence of a witness tree in the access graph. The proofs hold with respect to a rather weak oblivious adversary that randomly permutes the arrival order of the messages in each round.

Allocation with retries. We introduce an algorithm, called RETRY, that parallelizes two rounds of the THRESHOLD algorithm and avoids sending heights and assigning priorities to the choices. A ball that is not accepted in the first round, randomly chooses a new bin in the second round and commits to it. We refer to such an incident as a *retry*. We note that using retries violates the topological assumption. We prove that the maximum load obtained by algorithm RETRY is $\Theta(\sqrt{\ln n / \ln \ln n})$.

Our analysis method is of separate interest. It is based on analyzing the expected number of retries and proving that the number of retries is concentrated around the expected value. This technique yields both upper and lower bounds. We remark that the lower bound can be proved similarly to the lower bound we prove for the THRESHOLD algorithm.

A practical algorithm and its simulation. The gap between the load of the greedy algorithm (i.e., $\log_2 \log_2 n$) and the load of a 2-round algorithm such as PGREEDY (i.e., $\sqrt{\log_2 n / \log_2 \log_2 n}$) becomes noticeable only for very large values of n (e.g., $n > 2^{1024}$). This raises the need for conducting experiments (i.e., simulations) with smaller values of n (e.g., $n \in \left[10^6, 8 \cdot 10^6\right]$) since the asymptotic analysis does not yield results for such values of n. Concentration results (such as Lemmas 3 & 7) that characterize such random processes further justify simulations.

We designed an algorithm, called H-RETRY, with 3 rounds and a single synchronization point in which a lot of non-topological information is communicated. Our experiments show that for $10^6 \leq n \leq 8 \cdot 10^6$ balls, the maximum load is 3-4. This meets the best sequential results of Azar et al. [3] and Voecking [6], and beats previous results (load 4-5) reported for parallel algorithms [1].

Organization. In Sect. 2 we overview the general lower bound proved in [1,2]. We show that the topological assumption does not hold for the PGREEDY and THRESHOLD algorithms. In Sect. 3 we prove lower bounds for the PGREEDY and THRESHOLD algorithms. In Sect. 4 we present an algorithm with retires and analyze its performance. In Sect. 5 we present an heuristic and compare its performance by simulations.

2 Reopening the Lower Bound

Adler et al. [1] and Berenbrink et al. [2] proved a lower bound on the maximum load achieved by randomized parallel balls and bins algorithms. If each ball selects a constant number of random bins, the lower bound states that, with constant probability, the maximum load is $\Omega(\sqrt[r]{\ln n / \ln \ln n})$, where: (i) r denotes the number of communication rounds and (ii) n denotes both the number of bins

and the number of balls. The lower bound is based on a reduction to a random (hyper)-graph model.

For simplicity, we focus on the case that each ball chooses two bins independently and uniformly at random (i.u.r.). An *access graph* G over the bins is associated with the random choices of the balls. For each ball b, the edge $e(b)$ connects the pair of bins $(u_1(b), u_2(b))$ chosen by b. Note that the access graph may have self-loops (if a ball chooses the same bin twice) and parallel edges (if two balls choose the same pair of bins).

The neighborhood $N_r(b)$ of $e(b)$ in G is the set of vertices and edges that can be reached from an endpoint of $e(b)$ by a path that contains at most $r-1$ edges.

In [1], a ball b is said to be *confused* if $N_r(b) \setminus \{e(b)\}$ consists of two isomorphic trees rooted at the endpoints of $e(b)$. In [2], it is additionally required that these two rooted trees are complete trees of degree T and height $r-1$. A monotonicity assumption is made in [2] stating that deleting balls does not increase the maximum load, hence, edges in $N_r(b) \setminus \{e(b)\}$ that do not belong to one of the trees may be deleted.

Adler et al. [1] denoted a complete rooted tree of degree T and height r by a (T, r)-tree. The analysis of the lower bound in [1,2] is based on the following theorem.

Theorem 1 ([1,5]). *Let $r \leq \log\log n$ and $T = O(\sqrt[r]{\ln n / \ln\ln n})$. The access graph G contains a (T, r)-tree with probability at least $1/2$.*

The analysis is based on the observation that if an edge $e(b)$ is incident to the root of a (T, r)-tree, then ball b is confused. The analysis proceeds by proving that the root of such a tree is likely to have a load at least $T/2$. This argument is based on the the assumption (formalized below) that a confused ball breaks the symmetry by committing to a bin with a fair coin flip.

Assumption 1. *The decision of a ball b is based only on the topology of $N_r(b)$ and on random bits. Moreover, if both "sides" of the neighborhood $N_r(b)$ are isomorphic complete rooted trees, then the ball b commits to an endpoint of $e(b)$ by flipping a fair coin.*

We emphasize that topology does not include names of balls and bins, and therefore these names do not affect the decisions (this is formalized by the symmetry requirement).

Interestingly, both the PGREEDY and THRESHOLD algorithms introduced and analyzed in [1] do not satisfy this assumption.

Proposition 1. *Algorithms* PGREEDY *and* THRESHOLD *do not satisfy Assumption 1.*

Proof. In algorithm PGREEDY each bin sends back a height to a requesting ball. In terms of the access graph, each vertex (i.e., bin) consecutively numbers the edges incident to it. A ball (possibly confused) commits to the bin that returned the lower height. Part (I) in Fig. 1 depicts a confused ball b that commits to a bin deterministically.

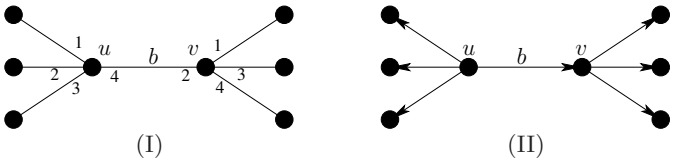


Fig. 1. Two violations of Assumption 1: (I) A confused ball b in PGREEDY chooses bin v since the height of b in v is 2. (II) A confused ball b in THRESHOLD chooses bin u because it is the first round's choice.

In algorithm THRESHOLD each ball sends a request together with its round number. Hence the edges of the access graph can be viewed as directed arcs (e.g, the tail of the arc is the bin chosen in round 1, and the head is the bin chosen in round 2). Algorithm THRESHOLD gives preference to the first round over the second round. Part (II) in Fig. 1 depicts a confused ball b that chooses a destination bin deterministically.

We remark that the C-Load Collision synchronous protocol [4,2] satisfies Assumption 1. Namely, less information is forwarded by the parties in the protocol, and hence, the lower bound holds for it.

Corollary 1. *The proof of the lower bound in [1] does not apply to algorithms* PGREEDY *and* THRESHOLD.

3 A Lower Bound for PGREEDY and THRESHOLD

Lower bound PGREEDY**.** Although Algorithm PGREEDY does not satisfy Assumption 1, we use Theorem 1 to prove a lower bound for algorithm PGREEDY. Since PGREEDY is a two-round algorithm, the witness tree is a $(T, 2)$-tree τ in the access graph G.

Lemma 1. *Let* $9 \leq T = O(\sqrt{\ln n / \ln \ln n})$. *The maximum load of the* PGREEDY *algorithm under an oblivious adversary is* $T/4 - 1$ *with constant probability.*

Proof. By Theorem 1 the access graph G contains a $(T, 2)$-tree τ with probability at least $1/2$. Conditioned on the existence of τ, we prove that root ρ of τ has load $T/4-1$ with probability at least $1/3$. Fix a ball b whose edge $e(b)$ is incident to the root, namely $e(b) = (\rho, v)$. Consider the heights h_ρ and h_v given to the requests of the ball b. The ball b commits to the root if $h_\rho < h_v$. Under an oblivious adversary, the heights of balls that request the same bin are a random permutation (with uniform distribution). Since both the root and v are of degree T, the probability that $h_\rho < h_v$ equals $1/2 - 1/(2T)$. Therefore, by linearity of expectation, the expected load of the root is at least $T/2 - 1/2$. The lemma follows from Markov's bound applied to T minus the load of the root.

Lower Bound THRESHOLD**.** Although Algorithm THRESHOLD does not satisfy Assumption 1, we use Theorem 1 to prove a lower bound for algorithm THRESHOLD. We first focus on a two-round version with threshold T, and assume that the access graph contains a $(T, 2)$-tree τ. Let $v_1, \ldots, v_T$ denotes the children of the root of τ.

Proposition 2. *The probability that at least $T/2$ balls are accepted by bin v_i in the first round is at least $1/2$.*

Proof. Each ball incident to v_i is randomly oriented, and the number of balls accepted by v_i in the first round equals the out-degree of v_i. With probability at least $1/2$, the out-degree of v_i is at least $T/2$.

Proposition 3. *The probability that one of the bins $v_1, \ldots, v_T$ accepts at least $T/2$ balls in the first round is at least $1 - 2^{-T}$.*

Proof. The loads of the bins $v_1, \ldots, v_T$ are independent since the sets of balls incident to each bin are disjoint. The proposition follows from Proposition 2.

THRESHOLD**: the case of $r > 2$ rounds.** Assume that the access graph contains a (T, r)-tree τ. Let $v_1, \ldots, v_{T\cdot(T-1)^{r-2}}$ denote the parents of the leaves of τ. The following proposition follows from Hoeffding's Inequality.

Proposition 4. *The probability that at least $T/(2r)$ balls are accepted by bin v_i in the first round is at least $1 - e^{-T/(2r^2)}$.*

Proposition 5. *The probability that one of the bins $v_1, \ldots, v_{T\cdot(T-1)^{r-2}}$ accepts at least $T/(2r)$ balls in the first round is at least $1 - e^{-T^2\cdot(T-1)^{r-2}/(2r^2)}$.*

The following corollary assumes an oblivious adversary.

Corollary 2. *Let $r \leq \log\log n$ and $T = O(\sqrt[r]{\ln n / \ln\ln n})$. With constant probability, the maximum load obtained by algorithm* THRESHOLD *with r rounds and threshold T is $T/2r$.*

4 A Tight Analysis of an Algorithm with Retries

In this sect. we consider an algorithm, called RETRY, that does not forward height information or associate preferences to the first two choices of each ball. To compensate for this limitation, rejected balls retry a third bin. Algorithm RETRY can be viewed as an attempt to parallelize the two rounds of the THRESHOLD algorithm. That is, all balls participate in the two rounds, and doubly rejected balls are given a third chance. Alternatively, RETRY can be viewed as an attempt to avoid forwarding heights (as in PGREEDY).

Organization. We begin by introducing the RETRY algorithm. Bounds on the number of rejected replicas and doubly rejected balls are proved in Sect. 4.2 and 4.3, respectively. The tight bound on the maximum load is proved in Sect. 4.4.

4.1 Algorithm RETRY: Description

Each ball is replicated twice, and each replica chooses a random bin. The algorithm is parametrized by a threshold T. Each bin accepts at most T replicas. A ball is *doubly rejected* if both its replicas are rejected. A doubly rejected ball chooses i.u.r. a new bin and commits to it.

Algorithm 1. RETRY *(threshold T, number of balls & bins n):*

1. Round 1*:*
   (a) *Each ball b generates two replicas. Each replica b' chooses i.u.r. a bin $u(b')$ and sends a request to the bin.*
   (b) *Upon receiving a request from replica b', if T replicas have been already accepted, then replica b' is rejected (i.e., a reject message is sent to ball b). Otherwise, the replica is accepted (i.e., an accept message is sent to ball b).*
2. Round 2:
   (a) *Each ball that receives two reject messages chooses i.u.r. a bin $u(b)$ and sends a commit message to b. (A commit message cannot be rejected. A ball that receives two accept messages may send a withdrawal message to one of the accepting bins.)*

Note that Algorithm RETRY is nonadaptive, as the bin choices (including the commit request) may be chosen before any communication takes place.

4.2 Analyzing the Number of Rejected Replicas

We begin by bounding the expected number of rejected replicas (Lemma 2). In the proof, we use linearity of expectation, Poisson approximations of the binomial distribution inequalities, and bound the tail of a Poisson distribution by a geometric series. The following lemma quantifies the intuition that the load in each bin is a Poisson random variable. Thus, the expected number of rejected replicas is approximately $n \cdot \Pr(load(bin) > T) \approx n \cdot \frac{2^{T+1}}{(T+1)!}$.

Notation. Suppose that m ball replicas are tossed into n bins i.u.r., and let X_i denote the number replicas in bin i. The number of replicas rejected by bin i equals $X_i - T$. Let $f(\boldsymbol{X})$ denote the number of rejected replicas. Then, $f(\boldsymbol{X}) = \sum_{i=1}^{n} \max(X_i - T, 0)$. See Fig. 2 for a depiction of the bin loads when $2n$ replicas are i.u.r. tossed in n bins, for $n = 8 \cdot 10^6$.

Lemma 2. *If the threshold T satisfies $6 \leq T \leq \sqrt{n}$, then*

$$e^{-5} n \cdot \frac{2^{T+1}}{(T+1)!} \leq \mathbb{E}[f(\boldsymbol{X})] \leq 2n \cdot \frac{2^{T+1}}{(T+1)!} .$$

The following lemma states that the number of rejected replicas is concentrated around its expected value. The proof introduces a Doob martingale and applies the Azuma-Hoeffding inequality with the Lipschitz condition (similarly to the analysis of the number of empty bins in [7]).

Lemma 3. $\Pr\left(\left|f\left(\boldsymbol{X}\right)-\mathbb{E}\left[f\left(\boldsymbol{X}\right)\right]\right|\geq\varepsilon\right)\leq 2\cdot e^{-\varepsilon^2/n}$.

Proof. We denote the replicas by $1\leq\beta\leq 2n$. Let ξ_β denote the bin of replica β. The random variables $\{\xi_\beta\}_{\beta=1}^{2n}$ are independent and uniformly distributed. Define $\tilde{f}$ so that $\tilde{f}\left(\boldsymbol{\xi}\right)=f\left(\boldsymbol{X}\right)$. Let $Z_0=\mathbb{E}\left[\tilde{f}\left(\boldsymbol{\xi}\right)\right]$ and $Z_k=\mathbb{E}\left[\tilde{f}\left(\boldsymbol{\xi}\right)\mid\xi_1,\xi_2,...,\xi_k\right]$. The sequence $Z_0,Z_1,...$ is a *Doob martingale* [7].
Note that $Z_{2n}=\mathbb{E}\left[\tilde{f}\left(\boldsymbol{\xi}\right)\mid\xi_1,\xi_2,...,\xi_{2n}\right]=\tilde{f}\left(\boldsymbol{\xi}\right)$. The function $\tilde{f}$ satisfies the Lipschitz condition with bound $c=1$. We apply the Azuma-Hoeffding inequality with the Lipschitz condition, namely, $\Pr\left(\left|Z_{2n}-Z_0\right|\geq\varepsilon\right)\leq 2\cdot e^{-\varepsilon^2/n}$, and the lemma follows.

Corollary 3. $\Pr\left(\left(f\left(\boldsymbol{X}\right)-\mathbb{E}\left[f\left(\boldsymbol{X}\right)\right]\right)^2\geq\gamma\right)\leq 2\cdot e^{-\gamma/n}$.

We use Coro. 3 to prove a linear bound on the variance of the number of rejected replicas. This bound is a multiplicative constant above the variance of the sum n independent binomial $B(2n,1/n)$ random variables.

Lemma 4. $Var\left(f\left(\boldsymbol{X}\right)\right)\leq 4n$.

4.3 Analyzing the Number of Doubly Rejected Balls

Let Y denote random variable that equals the number of doubly rejected balls. The following lemma bounds the expected number of doubly rejected balls. We use a conditioning on the number of rejected replicas and Lemma 4 to show that the expected number of doubly rejected balls is $\Theta\left(\frac{1}{n}\cdot\mathbb{E}^2[f(\boldsymbol{X})]\right)$. To simplify notation let $H(n)\triangleq\sqrt{\ln n/\ln\ln n}$.

Lemma 5. $\frac{\mathbb{E}^2[f(\boldsymbol{X})]}{4n}-1\leq\mathbb{E}\left[Y\right]\leq\frac{\mathbb{E}^2[f(\boldsymbol{X})]}{4n}+1$.

Lemma 6. *If* $\ln\ln n\leq T\leq H(n)$*, then, for* n *sufficiently large,* $\Omega(\frac{n^{3/4}\cdot\ln\ln n}{\ln n})<\mathbb{E}\left[Y\right]<\frac{n}{\ln n}$.

The following lemma shows that the number of doubly rejected balls is concentrated around its expected value. The proof is similar to the proof of Lemma 3.

Lemma 7. $\Pr\left(\left|Y-\mathbb{E}\left[Y\right]\right|\geq\varepsilon\right)\leq 2\cdot e^{-\varepsilon^2/n}$.

Corollary 4. *Let* $\ln\ln n\leq T\leq H(n)$*. The following equations hold with probability at least* $1-\frac{2}{n}$*:*

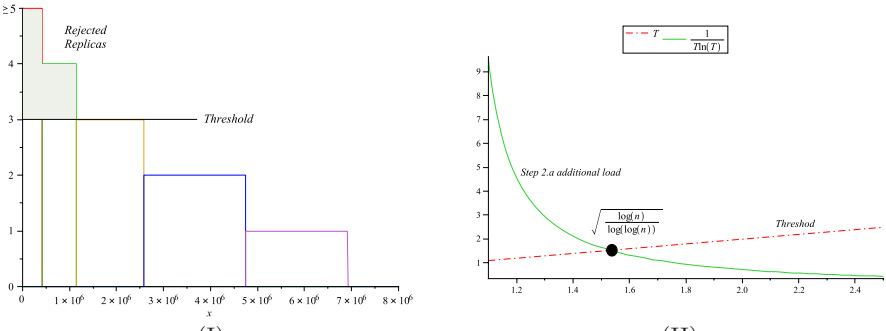
1. $\left|Y-\mathbb{E}\left[Y\right]\right|\leq\sqrt{n\ln n}$.
2. $Y=\Theta\left(\frac{1}{n}\cdot\mathbb{E}\left[f\left(\boldsymbol{X}\right)\right]^2\right)$ *(if* n *is sufficiently large).*
3. $Y<(1+o(1))\cdot\frac{n}{\ln n}$ *(if* n *is sufficiently large).*

4.4 Putting It All Together

The maximum load obtained by Algorithm RETRY is attributed to two factors: at most T replicas accepted in the first round and the load caused by the retries of the double rejected balls.

Theorem 2 ([8,9]). *If m balls i.u.r. each choose one ball out of n, then w.h.p. the maximum load equals $\Theta\left(\frac{\ln n}{\ln\left(1+\frac{n}{m}\cdot\ln n\right)}+\frac{m}{n}\right)$.*

By Theorem 2, if $T \leq \frac{\ln n}{\ln \ln n}$, then w.h.p. at least one bin accepts T replicas. The load caused by the retires of the doubly rejected balls is bounded again using Theorem 2. These two factors are balanced below to minimize the asymptotic maximum load (see Fig. 2 for a depiction of the trade-off).



(I) (II)

Fig. 2. (I) The distribution of bin loads in an experiment with $n = 8 \cdot 10^6$ bins and $16 \cdot 10^6$ ball replicas. The x-axis depicts the bins in descending load order. The y-axis depicts the load of each bin. The leftmost bar represents bins with load 5 or higher. (II) The trade-off between the threshold T and the additional load caused by the retries in Step 2a. The x-axis denotes T, and the y-axis denotes the value of T and $1/(T\ln(T))$.

Theorem 3. *Let $\ln \ln n \leq T \leq H(n)$, then w.h.p. the maximum load obtained by Algorithm* RETRY *is $\Theta\left(T+\frac{\ln n}{T\cdot\ln T}\right)$.*

Proof. Let L_i denote the maximum load incurred by round i, and let L denote the final maximum load. By Theorem 2, w.h.p. $L_1 = T$. We now prove that $L_2 = \Theta\left(\frac{\ln n}{T\cdot\ln T}\right)$.

Corollary 4, and Lemma 2 imply that:

$$Y = \Theta\left(\frac{1}{n}\cdot\mathbb{E}\left[f\left(\boldsymbol{X}\right)\right]^2\right) = \Theta\left(n\cdot\left(\frac{2^{T+1}}{(T+1)!}\right)^2\right). \tag{1}$$

By Coro. 4 w.h.p. $Y < (1+o(1))\cdot\frac{n}{\ln n}$. By Theorem 2 w.h.p. $L_2 = \Theta\left(\frac{\ln n}{\ln(n/Y)}\right)$. Plugging in Eq. 1 gives $L_2 = \Theta\left(\frac{\ln n}{T\cdot\ln T}\right)$, as required. The upper bound now follows since $L \leq L_1 + L_2$. The lower bound follows from $L \geq \max\{L_1, L_2\} \geq (L_1+L_2)/2$, and the theorem follows.

Corollary 5. *Let* $\ln\ln n \leq T \leq H(n)$*, then the maximum load obtained by Algorithm* RETRY *is minimized for* $T = \Theta(H(n))$*, and for* $T = \Theta(H(n))$ *the maximum load is w.h.p.* $\Theta(H(n))$*.*

Lemma 8. *If* $T < \ln\ln n$ *or* $H(n) < T$*, then w.h.p. the maximum load obtained by Algorithm* RETRY *is* $\Omega(H(n))$*.*

Proof. If $T < \ln\ln n$ then the number of doubly rejected balls (e.g. Y) increases, thus increasing the additional load that Step 2a incurs.

If $H(n) < T$, then by Theorem 2 w.h.p. the maximum load in the first round is at least $\min\left\{T, \frac{\ln n}{\ln\ln n}\right\}$. Since $\frac{\ln n}{\ln\ln n} = H^2(n)$, w.h.p. the maximum load is $\Omega(H(n))$.

5 Algorithm H-RETRY

Description. The algorithm is a 3-round algorithm and has a threshold parameter T. The first round is identical to PGREEDY. In the second round, each ball forwards the heights of one replica to the bin of the other replica. Namely, replica heights are forwarded between bins at distance 2 in the access graph.

A synchronization point is defined at this stage, namely, each bin must receive all its requests and the heights of the siblings of the replicas requesting the bin. Let bin_u denote the set of replicas that requested bin u. Each bin now partitions its set bin_u into 3 parts, A_u, SD_u, ED_u, where A_u is the set of accepted replicas, SD_u is the set of rejected replicas due to safe deletes, and ED_u is the set of rejected replicas due to excess deletes.

The subset SD_u is defined as follows. For each replica $b' \in bin_u$, let b'' denote its sibling replica. Let $\delta_{b'} \triangleq h(b') - h(b'')$. Note that $\delta_{b'}$ equals the difference between the local height of replica b' and the height of its sibling. Let $|bin_i|$ denote the cardinality of bin_i. Sort the replicas in bin_u in ascending height order. The set SD_u consists of the suffix of bin_u containing $\max\{0, |bin_u| - T\}$ replicas. (SD_u is empty if $|bin_u| \leq T$.)

The subset ED_u is defined as follows. Note that, if $|bin_u \setminus SD_u| > T$, then every replica $b' \in bin_u \setminus SD_u$ satisfies $\delta_{b'} \leq 0$. Sort the replicas in $bin_u \setminus SD_u$ in descending $\delta_{b'}$ order. Break ties by smallest height first. The set ED_u consists of the prefix of $bin_u \setminus SD_u$ consisting of $\max\{0, |bin_u \setminus SD_u| - T\}$ replicas. (ED_u is empty if $|bin_u \setminus SD_u| \leq T$.)

The subset A_u consists of the remaining replicas, namely $A_u = bin_u \setminus (SD_u \cup ED_u)$. Each bin u sends reject messages to all balls whose replicas are in $SD_u \cup ED_u$ and accept message to balls whose replicas are in A_u.

In the third round, upon receiving accept/reject messages for both replicas, each ball proceeds as follows. If both siblings are accepted, then the ball sends a withdraw message to the bin with the higher load (break ties arbitrarily). If both replicas were rejected, then the ball retries by i.u.r. choosing two random bins. These bins accept only if accepting the new ball does not overload the bin.

Discussion. Algorithm H-RETRY satisfies the requirements of the model described in Sect. 1.1, except for having one synchronization point.

Table 1. Results for bin load frequencies, number of retries, and rejection frequencies in 50 trials per four values of n ranging from 1 million to 8 million. For each bin load, the frequencies obtained in the trials is presented by the median and half the difference between the maximum and minimum frequency.

n	*T*	*Bin Loads*					*#Retries*	*Rejection frequencies*				
		0	*1*	*2*	*3*	*4*		*0*	*1*	*2*	*3*	*>=4*
1M	3	201975.5 ±512	601483.5 ±1028.5	191069.5 ±608	5485 ±129	0 ±0	38 ±10.5	44	6	0	0	0
2M	3	404052.5 ±784.5	1202875.5 ±1434	382162.5 ±721.5	10985.5 ±255.5	0 ±0	76 ±31	36	14	0	0	0
4M	3	808048.5 ±1089.5	2405691.5 ±2131	764304.5 ±998	21931.5 ±259.5	0 ±0	149 ±27	26	19	4	1	0
8M	3	1616149.5 ±1387.5	4811338.5 ±2758	1528371 ±1466	43972.5 ±397	0 ±0	310 ±44.5	24	15	8	1	2
8M	4	1596627.5 ±1379	4840281 ±2671.5	1529629 ±1487.5	33367.5 ±394.5	87 ±21.5	0 ±1	50	0	0	0	0

Our experiments are, of course, synchronous. This leads to a "layering" phenomenon since two siblings are more likely to receive the same height. One could shuffle the heights in the simulation and obtain slightly better results. We did not shuffle heights, so the layering phenomenon had a slight adverse effect.

There are a many other ways to deal with doubly rejected balls. First, since they are so few, one could simply have each such ball choose a random bin. Since there are so few such balls, they incur only a constant additional load. This is perhaps the simplest solution. A second option is to reserve a small portion of the bins for retries so that in the first round the reserved bins are not chosen.

Duplicate siblings due to retries can be removed by adding a fourth round. Namely, in the second half of the third round send accept and reject messages so a ball can send withdraw messages in round 4 to eliminate duplicates. We emphasize that the complications caused by retries are due to very few balls, hence, it is not clear that these issues are of practical interest.

Experimental results. We conducted experiments for $10^6 \leq n \leq 8 \cdot 10^6$. For each n, the results for 50 trials are presented in Table 1. The value of the threshold was $T = 3$ in all cases, except for $n = 8 \cdot 10^6$, where we also used $T = 4$ (last row). The frequencies of the bin loads are presented. For example, a load of zero means the bin is empty. For each load, the range of frequencies in the experiments is given by the median and half the difference between the maximum and the minimum frequency. Note that the load frequencies are sharply concentrated. The column labeled #Retries contains the number of balls that are doubly rejected, and therefore, required a retry. Note that the number of doubly rejected balls roughly doubles as n doubles and is also sharply concentrated. The frequencies of the number of doubly-rejected balls that remain rejected at the end appear in the last 4 columns. We never encountered more than 5 balls that were not finally accepted.

Our experiments show that, even for 8 million balls, only a handful of balls are finally rejected. One could reassign them, if needed, using an extra round.

Alternatively, one could use three choices in the retry stage instead of two or simply accept the retries while increasing the maximum load only by one with high probability.

Acknowledgments

We thank Michael Mitzenmacher for helpful correspondence. We thank Haim Kaplan, Dana Ron and Boaz Patt-Shamir for their suggestions, and the audience of the Algorithms Seminar in the School of Computer Science in Tel-Aviv University for their remarks and feedback.

References

1. Adler, M., Chakrabarti, S., Mitzenmacher, M., Rasmussen, L.E.: Parallel randomized load balancing. Random Struct. Algorithms 13(2), 159–188 (1998)
2. Berenbrink, P., auf der Heide, F.M., Schröder, K.: Allocating Weighted Jobs in Parallel. Theory of Computing Systems 32(3), 281–300 (1999)
3. Azar, Y., Broder, A., Karlin, A., Upfal, E.: Balanced allocations. SIAM journal on computing 29(1), 180–200 (2000)
4. Stemann, V.: Parallel balanced allocations. In: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, pp. 261–269. ACM, New York (1996)
5. Czumaj, A., auf der Heide, F., Stemann, V.: Contention Resolution in Hashing Based Shared Memory Simulations. SIAM Journal On Computing 29(5), 1703–1739 (2000)
6. Voecking, B.: How Asymmetry Helps Load Balancing. Journal of the ACM 50(4), 568–589 (2003)
7. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
8. Kolchin, V., Sevastyanov, B., Chistyakov, V.: Random Allocations. John Wiley & Sons, Chichester (1978)
9. Raab, M., Steger, A.: "Balls into bins" - a simple and tight analysis. In: Rolim, J.D.P., Serna, M., Luby, M. (eds.) RANDOM 1998. LNCS, vol. 1518, pp. 159–170. Springer, Heidelberg (1998)

An Improved Strategy for Exploring a Grid Polygon*

Agnieszka Kolenderska[1], Adrian Kosowski[1,2], Michał Małafiejski[1], and Paweł Żyliński[3]

[1] Dept of Algorithms and System Modeling, Gdańsk University of Technology
Narutowicza 11/12, 80233 Gdańsk, Poland
{agnieszka,adrian,mima}@kaims.pl
[2] LaBRI - Université Bordeaux 1 - CNRS
351, cours de la Liberation, 33405 Talence, France
kosowski@labri.fr
[3] Institute of Computer Science, University of Gdańsk
Wita Stwosza 57, 80952 Gdańsk, Poland
pz@inf.univ.gda.pl

Abstract. We study the problem of exploring a simple grid polygon using a mobile robot. The robot starts from a location which is adjacent to the boundary of the polygon, and after exploring all the squares, has to return to its starting location. The robot is equipped with memory, but has no prior knowledge of the explored terrain. The view of the terrain is restricted to the four squares directly adjacent to the robot's current location. The performance of the exploration strategy is measured in terms of the competitive ratio, with respect to the length of the optimal path for an exploration with complete knowledge of the terrain.

We propose a new exploration strategy which achieves a competitive ratio of 5/4, whereas the previously best approach [Icking, Kamphans, Klein, and Langetepe; *Proc. COCOON'05*] has a competitive ratio of 4/3. The analysis for our algorithm is tight. Moreover, we show that no exploration strategy is ever better than 20/17-competitive, thus improving the previous lower bound of 7/6.

Keywords: Exploration problem, Path planning, Mobile robot, Grid polygon, Competitive ratio.

1 Introduction

In this paper we investigate one of the basic path planning problems for a mobile robot, namely that of exploring the area of an unknown polygon. This type of geometric problem has received a lot of attention, both in a continuous and a discrete setting. In the continuous variant, sometimes referred to as the *milling problem* [2] or the *mobile robot covering problem* [5], the robot has non-zero spacial dimensions

* The research was partially funded by the KBN Grant 4 T11C 047 25, by the ANR project "ALADDIN", and by the INRIA project "CEPAGE".

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 222–236, 2010.

and moves along a continuous trajectory, with the goal of covering each point of the explored environment at least once. Herein, we focus on the discrete variant of the problem, defined for polygons composed of unit cells; this can in fact also be regarded as a special case of a graph exploration problem.

The explored environment is assumed to be a simple two-dimensional orthogonal polygon with integer coordinates. The environment is divided into unit squares (called *cells*), with coordinates of corners belonging to the integer grid. The robot is always assumed to occupy the whole of a single cell. The robot's motion is restricted to one of four directions: East, West, North or South. Time is measured in discrete steps, and in one step, the robot can only move from one cell to a cell which is directly adjacent along a side (at a distance of one). The task of the robot is to perform an exploration of all the cells of the polygon and return to the cell of its initial location. The robot can only proceed through cells which are located within the explored polygon, and the starting cell is assumed to be adjacent to the boundary of the polygon.

In such a scenario, one can define different limitations to the robot's capabilities: its knowledge of the polygon, memory, ability to leave markers in the terrain. In this paper we assume that the robot is equipped with sufficient memory and processing power, but has no prior knowledge of the explored polygon. Upon entering a cell and before deciding on the next move, the robot has only a sense of the direction of its current heading (a compass), and a local view of the surroundings, indicating which of the four cells directly adjacent to its current location belong to the polygon. Thus, the next move in the exploration has to be computed based on the fragment of the map of the terrain which the robot has already discovered; in some way, the behaviour of the robot resembles an on-line algorithm. Our goal is to design the best possible strategy for the robot, and to compare it to the optimal strategy for the off-line version of the problem, i.e., the scenario in which the robot is initially given a map of the entire terrain, together with a marker representing its initial location.

Related work. The off-line version of the problem can be rephrased in terms of the unweighted Travelling Salesman Problem (TSP) on a special class of graphs. The input graph is then some (planar) subgraph of the two-dimensional grid, with nodes defined as cells of the polygon and edges connecting each cell with the four cells neighbouring along a side. This problem was shown to be NP-hard for polygons with holes by Itai, Papadimitriou and Szwarcfiter [12]; the hardness of the problem for polygons without holes remains an open question. On the other hand, since the considered TSP instance is planar and has a natural metric, the problem admits a Polynomial-Time Approximation Scheme, using the techniques of Arora [3], Grigni, Koutsoupias, and Papadimitriou [8], or Mitchell [14]. These results hold regardless of whether the polygon is allowed to have holes or not. The case without holes also admits some simple and efficient approximation algorithms, for example the linear-time 6/5-approximation of Arkin, Fekete, and Mitchell [2].

For the case of a robot having only a local view of the adjacent cells, a simple exploration is achieved by performing a DFS exploration of the cells. In general,

Table 1. Bounds on the cover length of an unknown polygon for a robot with local view (S_{OPT} – length of the optimal cover path with full knowledge of the polygon, C – polygon area, B – number of boundary cells, E – perimeter of the polygon, W – sinuosity of the polygon [13], H – number of holes, c – an additive constant)

Type of polygon	Upper bound	Ref.	Lower bound	Ref.
With holes	$2C - 2$	DFS	$2\,S_{OPT} - c$	[5,10]
	$B + C$	SpiralSTC [5]		
	$C + \frac{1}{2}E + W + 3H - 2$	CellExplore [10]		
Without holes	$\frac{4}{3}\,S_{OPT}$	SmartDFS [11]	$\frac{7}{6}\,S_{OPT} - c$	[11]
	$\frac{5}{4}\,S_{OPT}$	**Thm. 2**	$\frac{20}{17}\,S_{OPT} - c$	**Thm. 3**

such a route is a 2-competitive solution to the TSP problem, since it traverses twice each of the edges of the spanning tree of the cell graph. In fact, for polygons with holes such an approach is essentially the best possible: it is known [5,10] that there does not exist a strategy which achieves a competitive ratio better than 2. However, some strategies for covering polygons with holes were constructed by Gabriel and Rimon [5], and Icking et al. [10]. In these approaches, the length of the exploration is bounded in terms of the number of holes, area, perimeter, and certain other parameters of the polygon (see Table 1); hence, in some cases the performance of these strategies is better than that of DFS. For the case when the covered polygon has no holes, Icking et al. [11] put forward a modification of the DFS approach which they called SmartDFS. SmartDFS was shown in [11] to be a 4/3-competitive algorithm, and it was also proved that there does not exist an approach to the covering problem which is better than 7/6-competitive.

Recently, Herrmann, Kamphans, and Langetepe [9] have investigated the related problem of exploration with local view of polygons on the triangular and hexagonal grids, i.e., in settings where a cell is a triangle or a hexagon. They showed that there does not exist an algorithm which is better than 7/6-competitive for a triangular polygon, and better than 13/11-competitive for a hexagonal polygon. They also proposed a modified variant of SmartDFS for hexagonal and triangular polygons. The corresponding off-line versions of the problem, for a robot with full knowledge of the map in a triangular or hexagonal grid polygon, are known to be NP-hard [1,7].

Contribution and outline of the paper. The paper is organised as follows. In Section 2 we propose a new strategy with local view for exploring orthogonal grid polygons without holes, and prove that it achieves a competitive ratio of 5/4, thus improving the ratio of 4/3 achieved by the SmartDFS approach. We

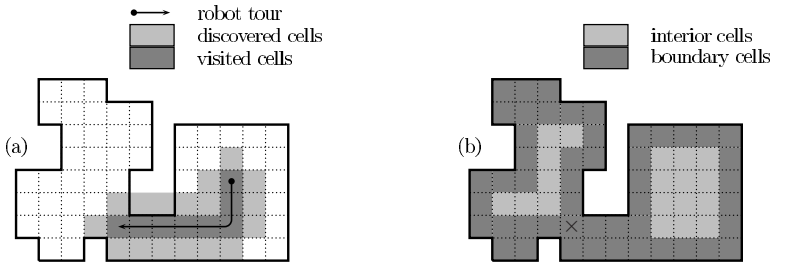


Fig. 1. (a) Discovered and visited cells. (b) Interior and boundary cells. The unique split cell is marked with a cross (×) .

show that the analysis of the competitive ratio of our algorithm is tight. In Section 3 we provide a new lower bound of 20/17 on the competitive ratio of any exploration algorithm with local view for the studied class of polygons. Some concluding remarks are made in Section 4.

Notation. We divide the polygon P and the surrounding plane into unit squares (cells). Each cell of the polygon is either not adjacent to the boundary of the polygon, or has at least one corner lying on the boundary. This divides the set of cells of the polygon into two disjoint sets: the *skeleton* (*interior cells*) and the *boundary* cells (Fig. 1(b)). We call a cell *discovered* if at least one of its neighbors has already been visited by the robot (Fig. 1(a)).

We will say that a cell $x \subset P$ is a *split* cell of a simple polygon P, if cell x is encountered more than once in a cyclic enumeration (clockwise traversal) of the boundary cells of the polygon; see Fig. 1(b) for an example. Note that $P \backslash x$ is a union of simple polygons; we will call the minimal number of disjoint simple polygons which cover polygon $P \backslash x$ the *multiplicity* of the split cell x. We define the *split number* of the polygon as $T = T_2 + 2T_3 + 3T_4$, where T_i is the number of split cells with multiplicity i, for $i = 2, 3, 4$.

The total number of cells of the polygon P is denoted by $C(P)$, or simply C if this does not lead to misunderstanding. We denote the length of the optimal length of a solution which can be achieved by a robot with full knowledge (a map) by S_{OPT}, and the length of the robot's path computed by algorithm A by S_A. We say that exploration algorithm A is α-competitive if there exists a constant c such that for any polygon P we have $S_A(P) \leq \alpha S_{OPT}(P) + c$.

Throughout the paper we will assume that the robot is located in a corner cell of the polygon, and that the initial heading of the robot is such that the cell behind it and the cell directly to the left are outside the polygon, and the cell in front of the robot is within the polygon. It is straightforward to show that for the considered algorithms the starting cell can be moved to an arbitrary cell adjacent to the boundary, without affecting the competitive ratio (only the additive constant c).

2 A 5/4-Competitive Algorithm

Our approach to the problem relies on an extensive modification of the Depth First Search (DFS) strategy. A slightly extended implementation of DFS could work as follows: the robot maintains a stack of unvisited cells. At every step, it adds to unvisited the cells which are adjacent to the robot's location. Visited cells are purged from the unvisited stack, and the robot then proceeds along the shortest path to the top-most cell of the stack, if the stack is non-empty. If the stack is empty, this means that the whole polygon has been explored, and the robot returns to its starting location. This is also the general idea behind the SmartDFS approach put forward in [11]. In fact, such an exploration strategy is shown to achieve a competitive ratio of 4/3, as long as the adjacent cells are pushed onto the unvisited stack in the following order: first the cell to the right, then the cell in front of, and finally the cell to the left of the robot (with respect to its current heading). Intuitively, the robot will thus traverse the unvisited boundary cells of the polygon in the clockwise direction, as long as this is possible, and when some sub-polygon consisting of unvisited cells has been completely surrounded by the robot's path, the robot first embarks upon a sub-exploration of this polygon. A worst-case example for SmartDFS is an exploration of the rectangular polygon of dimensions $3 \times n$ [11] in which this strategy will, in particular, visit twice $n-3$ of the cells belonging to the skeleton of the polygon.

In order to avoid such a problem and to achieve a competitive ratio of 5/4, in our strategy, we modify the DFS approach by adding a special rule set for handling situations in which the polygon intuitively "narrows down to a width of 3". The proposed strategy will behave differently from SmartDFS when one of the cells adjacent to the robot's location is identified as either a so-called *dead-end* or a *bottle-neck*, and such cells will be visited first. Both these notions are formally defined below (see Fig. 2 for a simple example displaying the general idea of the modifications).

Definition 1. *A cell c is called a* dead-end *at a given step of exploration if all of the following conditions are jointly fulfilled:*

- *Cell c has not yet been visited by the robot and it is adjacent along a side to the cell where the robot is currently located.*
- *Exactly* 3 *of the cells adjacent to c along a side have already been visited by the robot.*
- *There does not exist any cell c' outside of polygon P, such that c' has already been discovered by the robot and c' is adjacent to c along a side or across a corner.*

Definition 2. *A cell c is called a* bottle-neck *at a given step of exploration if all of the following conditions are jointly fulfilled:*

- *Cell c has not yet been visited by the robot and it is adjacent along a side to the cell where the robot is currently located.*
- *The union of the set of already explored cells and cell c surrounds all the sides of the outline of some polygon $P' \subset P$, such that P' consists of unvisited*

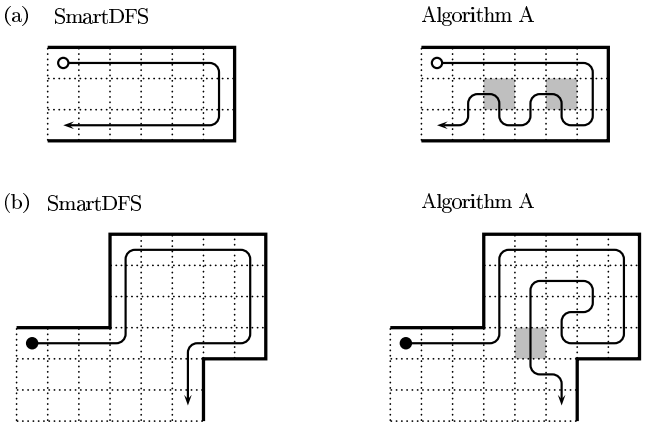


Fig. 2. Differences between our approach (Algorithm A) and SmartDFS [11] when handling: (a) dead-ends, (b) bottle-necks

cells, only, and P' contains the cell adjacent to c along the right side of c (for directions as understood when entering c from the robot's current location).
- *There does not exist any cell c' outside of polygon P', such that c' has already been discovered by the robot and c' is adjacent to c along a side.*
- *The cell located directly in front of the robot's location (with respect to its current heading) is located within the polygon.*

Observe that the definitions of a dead-end and a bottle-neck are such that the robot can always verify whether a given cell is a dead-end or a bottle-neck, based only on its current local view and its history of exploration.

2.1 The Algorithm

Our approach is defined precisely as follows. Throughout execution, the robot maintains a stack of cells called unvisited (as in DFS) and an additional stack called priority, used to implement the modified rules applied when encountering a dead-end or a bottle-neck. Both these stacks are initially empty. At each step of exploration, the precise procedure applied by the robot is given in the form of Algorithm A. In the pseudocode, the rules marked with the symbol (*) denote those, which are an extension with respect to the standard DFS approach. The robot terminates exploration when it re-enters its initial location, and the unvisited stack is empty.

2.2 Analysis of the Competitive Ratio

Observe that for any polygon, the optimal length of the off-line cover path fulfills the bound $S_{OPT} \geq C + T$. This is because in any exploration, each of the C cells of the polygon has to be visited at least once, and each of the split cells has to be visited more times, depending on the multiplicity of the split cell.

Algorithm A. Procedure for determining the next move of the robot.

At each time step do:

1. Update the stacks of unvisited and priority cells:
   - Push on top of the unvisited stack the 3 adjacent cells which are directly to the right of the robot, in front of the robot, and to the left of the robot (with respect to its current heading), in precisely that order, i.e., with the left cell coming topmost.
   - Search through the unvisited stack and purge from it all those cells, which have already been visited by the robot.
   - (*) Search through the priority stack and purge from it all those cells, which have already been visited by the robot.
2. Set the destination of the robot:
   - (*) If the priority stack is non-empty, then set: destination := top of the priority stack.
   - If the priority stack is empty, then:
     - If the unvisited stack is non-empty, then set: destination := top of the unvisited stack.
     - Otherwise, set: destination := initial location of the robot
3. (*) If the current location of the robot is adjacent to a dead-end or a bottle-neck, then alter the destination of the robot:
   - Push the current destination on top of the stack of priority cells.
   - Set destination := location of the adjacent dead-end/bottle-neck. If there is more than one dead-end or bottle-neck directly adjacent to the robot, we give preference to the one to the right of the robot, then the one in front of the robot, finally the one to the left of the robot.
4. Perform a move of the robot:
   - If the current destination is not a bottle-neck, move the robot to an adjacent cell which is closer to the destination than the robot's current location (with respect to the shortest path metric within the set of cells already discovered by the robot). If more than one such cell exists, break ties by arbitrarily choosing a cell which has not yet been visited.
   - (*) Otherwise, move the robot to its destination in the minimum number of steps, so as to cover on the way the entire sub-polygon P' (the unvisited sub-polygon surrounded by previously visited cells and the bottle-neck cell destination). The optimal route can always be determined, since the boundary of the covered sub-polygon is known.

Hence, we confine ourselves to bounding the length of path S_A with respect to $(C + T)$. More precisely, to prove that the approach given by Algorithm A has a competitive ratio of 5/4, we will establish the following theorem.

Theorem 1. *For any polygon P of area $C > 2$, we have $S_A \leq \frac{5C+5T-3}{4}$.*

Proof. First, observe that until a bottle-neck or dead-end are encountered, the algorithm always progresses along the boundary of the polygon. Thus we obtain a class of "narrow" polygons which are traversed optimally by Algorithm A.

Lemma 1. *If Algorithm A does not encounter any bottle-necks or dead-ends for polygon P, then $S_A = C + T$.*

To prove the theorem, let us now assume that P is a counter-example for our claim, i.e., a polygon such that $S_A(P) > \frac{5C(P)+5T(P)-3}{4}$, which is minimal in the following sense: out of all counter-examples, P has the minimal area of its skeleton, and of all such polygons, it has the minimal area $C(P)$. We will show through a sequence of lemmas that polygon P does not exist.

First, relying on the observation that a polygon with an empty skeleton cannot contain a 3×3 polygon as a sub-polygon, we obtain that P must have a non-empty skeleton; we omit the proof.

Lemma 2. *For any polygon P, such that the skeleton of the polygon P is empty and $C > 2$, Algorithm A covers P in $S_A \leq \frac{5C+5T-3}{4}$ steps.*

Consequently, we may assume that the skeleton of P is non-empty. Now, we consider the structure of the boundary cells of any polygon P. For a boundary cell c, let $dist(c) \geq 1$ denote the length of the shortest path from c to a nearest cell of the skeleton of P, according to the metric in which cells adjacent along sides or across corners are at a distance of 1 from each other (i.e., boundary cells adjacent to the skeleton along a side or across a corner have $dist(c) = 1$).

Note that the set of boundary cells of a polygon has an empty skeleton. We now observe that for our minimal counter-example P, the polygon cannot contain any boundary cells with $dist(c) > 2$. The proof uses simple local arguments (similar to those in the proof of Lemma 2) to show that otherwise there would exist a smaller counter-example P' to our claim; we again omit the details from this extended abstract.

Lemma 3. *For the minimal counter-example P, any boundary cell c fulfills $dist(c) \leq 2$.*

We now proceed to the main part of the proof. Taking into account the above lemma and the fact that there exists only a finite number of polygons having a given skeleton, subject to the condition $dist(c) \leq 2$ for any boundary cell c, we will now analyze the structure of polygon P by characterizing its skeleton, only.

A skeleton of a polygon will be called *elementary* if it forms a connected region of the plane (possibly across corners) and consists of at most 5 cells. There exist exactly 83 elementary skeletons which are distinct up to isometry. Taking into account Lemma 3, by an exhaustive computer search, we verify the following claim.

Lemma 4. *The minimal counter-example P does not have an elementary skeleton.*

To complete the proof of the theorem, we now show that there does not exist a minimum counter-example P which has a non-elementary skeleton. First, observe that if polygon P had no dead-ends and no bottle-necks, then it would not be a valid counter-example by Lemma 1. Likewise, supposing that P had no

bottle-necks, and the only dead-ends which appeared in the exploration were traversed as shown in Fig. 3(a). Then, we immediately obtain $S_A(P) \leq \frac{7}{6}C(P) + T(P)$, and once again P is not a valid counter-example. Hence, at some point during the exploration we must encounter a bottle-neck or a dead-end which is traversed differently than those from Fig. 3(a). Now, we apply a local replacement argument to show that the claim of the theorem holds for P if it holds for all polygons with a skeleton smaller than that of P. The part of the polygon in which we apply the replacement is determined by the earliest step during the exploration of P, at which any one of the following events occurs:

(1) A dead-end is encountered, and it is traversed differently than that shown in Fig. 3(a).
(2) A bottle-neck is encountered.
(3) A cell a is encountered, such that a is adjacent across a corner to the robot's location, and a is a split cell with respect to the set of unvisited cells of the polygon, see Fig. 3(b).

The proof is completed by analysing each of the Cases (1), (2), and (3); due to space constraints, we confine ourselves to a brief discussions of Case (1).

We can assume that P is a polygon with a skeleton size of n, and that the skeleton of P is not elementary. Then, next to the encountered dead-end (traversed differently than in Fig. 3(a)), the skeleton of polygon P must either include one of endings shown in Fig. 4, or else be disconnected.

We now attempt to cut off a part of polygon P to obtain a new polygon P', in such a way that the robot's tour within P' is the same as within P, except for a local modification. By an analysis of all possible cases, it can be shown that when considering an ending of the skeleton, we distinguish 20 types of local modifications which can be applied; 4 examples of such modifications are shown in Fig. 5. If the skeleton is disconnected, and these modifications cannot be applied, then one of the modifications from Fig. 6 can be applied instead.
Let us now compute the value of the following expression

$$\Delta = \Delta S - \frac{5}{4}(\Delta C + \Delta T) \tag{1}$$

where: $\Delta S = S_A(P) - S_A(P')$, $\Delta C = C(P) - C(P')$, $\Delta T = T(P) - T(P')$.

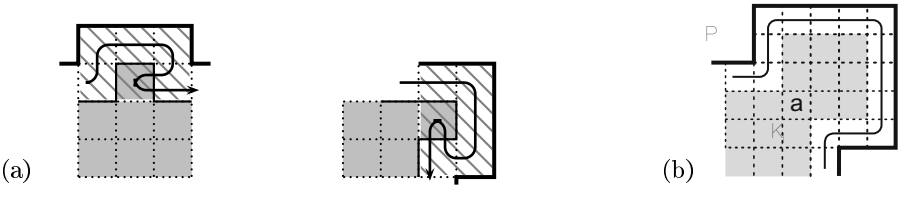


Fig. 3. Situations encountered during exploration: (a) special kinds of dead-ends, (b) split cell a of the set of unvisited cells (across a corner)

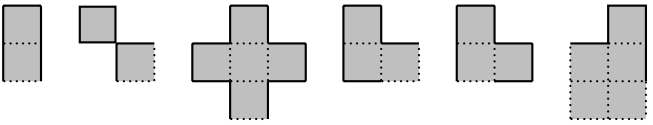


Fig. 4. Possible endings of the skeleton. Solid lines denote the outline of the skeleton.

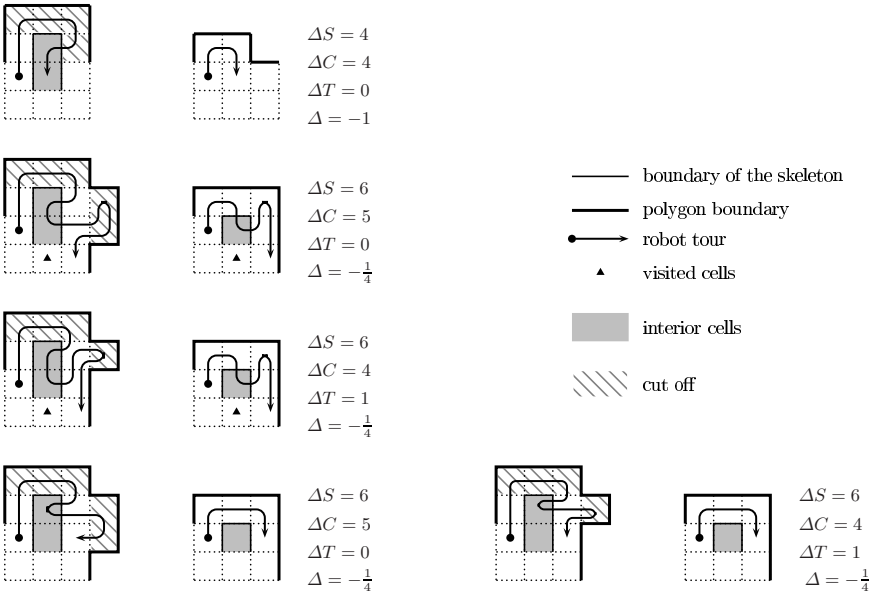


Fig. 5. Examples of reductions of endings of the polygon skeleton. Polygon P is shown in odd columns, the outcome P' of the modification is shown in even columns.

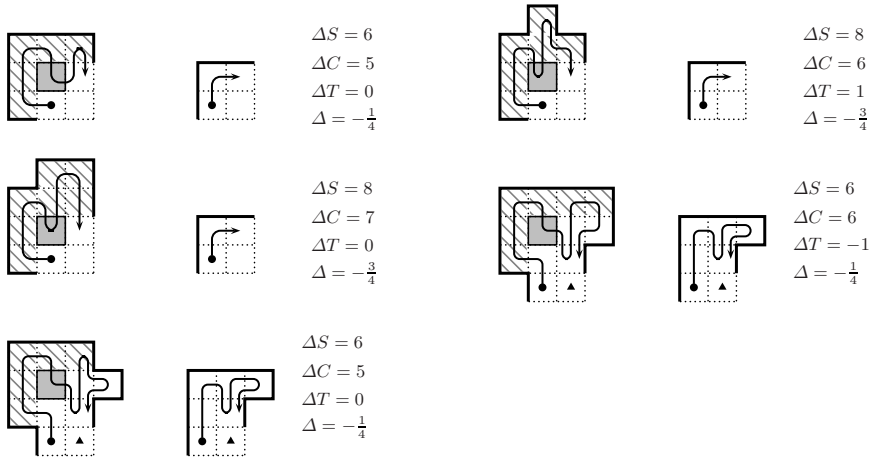


Fig. 6. Reductions for a disconnected skeleton of the polygon. Polygon P is shown in odd columns, the outcome P' of the modification is shown in even columns.

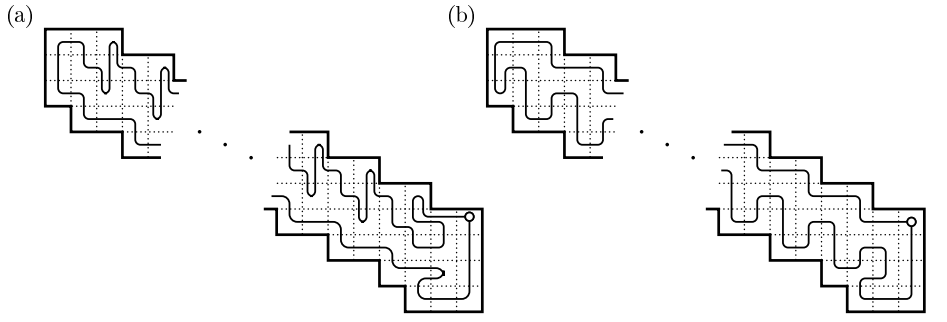


Fig. 7. A tight example for the competitive ratio: **(a)** The exploration performed by Algorithm A. **(b)** The optimal off-line covering path. The exploration starts from the cell marked with the circle.

Since P' has a smaller skeleton than P, and P was by assumption the minimal counter-example to the claim, we have:

$$S_A(P') \leq \frac{5C(P') + 5T(P') - 3}{4}. \tag{2}$$

Consequently:

$$S_A(P) \leq S_A(P') + \frac{5}{4}(\Delta C + \Delta T) \leq \frac{5}{4}C(P) - \frac{3}{4} + \frac{5}{4}T(P) = \frac{5C(P) + 5T(P) - 3}{4},$$

hence, P is not a counter-example to the claim.

Cases (2) and (3) are handled through a slightly different local replacement technique, which additionally relies on a decomposition of P into two polygons along a cut-line close to the encountered bottle-neck (or split cell). □

2.3 A Tight Example

Theorem 2. *The competitive ratio of Algorithm A is exactly* $5/4$.

Proof. The tight example is obtained by exploring the polygon shown in Fig. 7. The length of the optimal off-line tour is $S_{OPT} = C + 2$, and the length of robot's tour computed by Algorithm A is $S_A = \frac{5C-4}{4}$. The ratio of these two values precisely corresponds to the competitive ratio of the algorithm:

$$\frac{S_A}{S_{OPT}} = \frac{5C - 4}{4C + 8} \longrightarrow_{C \to \infty} \frac{5}{4}.$$

□

3 Lower Bound on the Competitive Ratio

Icking et al. [11] have shown that no exploration strategy of an orthogonal polygon is better than $\frac{7}{6}$-competitive. We obtain a tighter bound by substantially

extending their construction of the family of polygons which serves as a counter-example.

Theorem 3. *Let A be an exploration algorithm using a local view. If A is α-competitive, then $\alpha \geq \frac{20}{17}$.*

Proof. To prove the theorem, we construct a family of polygons $\mathcal{P}$ in such a way that, for any exploration algorithm E, there exists a polygon $P \in \mathcal{P}$ of arbitrarily large area $C(P)$, such that $S_E(P) \geq \frac{20}{17} S_{OPT}(P) - 2$.

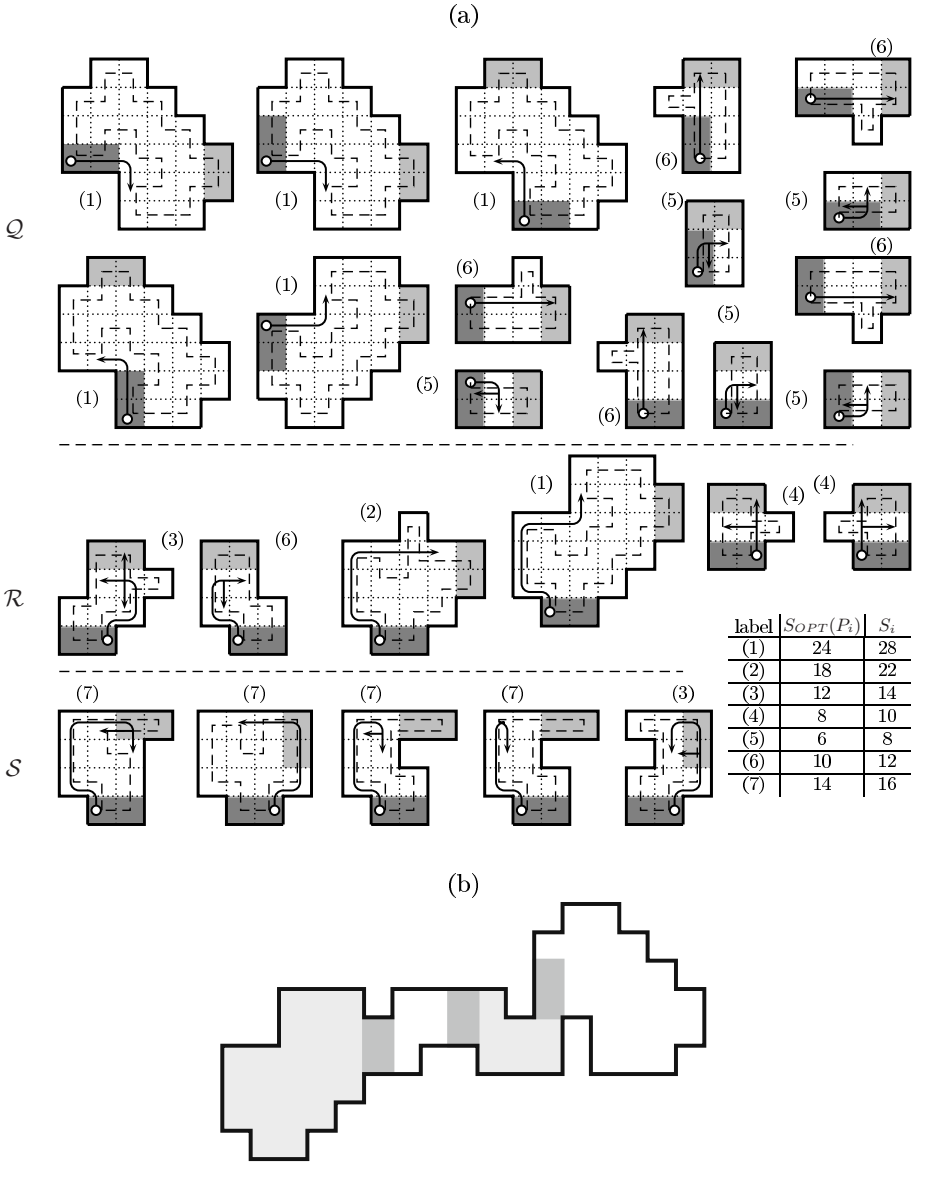
The considered family $\mathcal{P}$ is built up by connecting into a "chain" the elementary polygons from sets $\mathcal{Q}$, $\mathcal{R}$, $\mathcal{S}$, shown in Fig. 8. More precisely, we put $\mathcal{P} = \bigcup_{n \geq 1} \mathcal{P}_n$, and elements of the family $\mathcal{P}_n$, for all $n \geq 1$ are constructed from exactly n polygons from $\mathcal{Q} \cup \mathcal{R} \cup \mathcal{S}$ according to the approach described below. For $P \in \mathcal{P}_n$, we will write $P = (P_1, P_2, \ldots, P_n)$, where we have $P_i \in \mathcal{Q} \cup \mathcal{R} \cup \mathcal{S}$, and the following rules are applied:

- We have $P = P_1 \cup P_2 \cup \cdots \cup P_n$.
- The intersection $P_i \cap P_j$ is non-empty only for polygons adjacent in the sequence, i.e., when $|i - j| \leq 1$.
- The intersection $P_i \cap P_{i+1}$, for all $1 \leq i < n$, consists of exactly two adjacent cells of each of these polygons, as shaded in Fig. 8, located next to their North-East and South-West corners, respectively. Note that not every two polygons $P_i, P_{i+1} \in \mathcal{Q} \cup \mathcal{R} \cup \mathcal{S}$ can be put together so as to fulfill this condition.
- The following additional rules are fulfilled: $P_1 \in \mathcal{Q}$. Moreover, for all $1 \leq i < n$, we have $P_{i+1} \in \mathcal{Q}$ if and only if $P_i \in \mathcal{S}$.

Given any algorithm E, for any $n \geq 1$ we now construct by means of E a sub-optimal robot's route $\tau = (c_0, c_1, c_2, \ldots, c_S)$ covering some polygon $P = (P_1, P_2, \ldots, P_n) \in \mathcal{P}_n$; the details of the discussion are omitted due to space constraints. Here, c_t denotes the location of the robot after t steps of exploration, S is the length of the route, and $c_S = c_0$. Let S_i denote the length of route τ when restricted to the cells belonging to polygon P_i, i.e., the subsequence of τ obtained by removing all cells from outside P_i and then compacting identical adjacent elements. By the construction of polygon P, we have: $S \geq 2 + \sum_{1 \leq i \leq n} (S_i - 2)$. Moreover, the property of the constructed route τ is such that $S_i > S_{OPT}(P_i)$, with the minimal possible value of S_i for different types of polygons $P_i \in \mathcal{Q} \cup \mathcal{R} \cup \mathcal{S}$ listed in Fig. 8. Taking into account that for any polygon $P \in \mathcal{P}_n$ we have $S_{OPT}(P) = 2 + \sum_{1 \leq i \leq n} (S_{OPT}(P_i) - 2)$, we obtain:

$$\frac{S}{S_{OPT}(P)} \geq \frac{\sum_{1 \leq i \leq n} (S_i - 2)}{\sum_{1 \leq i \leq n} (S_{OPT}(P_i) - 2)} - O(1/n).$$

By finding the minimum possible value of the above expression for the values stated in Fig. 8, subject to the constraint that not more than $n/2$ of the polygons P_i belong to $\mathcal{S}$ (by the construction of family $\mathcal{P}_n$), we obtain the bound $\frac{S}{S_{OPT}(P)} \geq \frac{20}{17} - O(1/n)$. Such a bound is in fact asymptotically tight when we



label	$S_{OPT}(P_i)$	S_i
(1)	24	28
(2)	18	22
(3)	12	14
(4)	8	10
(5)	6	8
(6)	10	12
(7)	14	16

Fig. 8. Construction of polygons for the lower bound: (a) Sets of polygons $\mathcal{Q}$, $\mathcal{R}$, $\mathcal{S}$. Sets $\mathcal{R}$ and $\mathcal{S}$ also contain the respective polygons reflected with respect to the line North-East / South-West. (b) Example of a polygon belonging to $\mathcal{P}_4$.

have, for odd values of i, that $P_i \in \mathcal{Q} \cup \mathcal{R}$ with $S_i = 28$ and $S_{OPT}(P_i) = 24$, whereas for even values of i, $P_i \in \mathcal{S}$ with $S_i = 16$ and $S_{OPT}(P_i) = 14$. Note that: $\frac{(28-2)+(16-2)}{(24-2)+(14-2)} = \frac{20}{17}$. This completes the proof of the theorem. □

4 Final Remarks

We have presented a new exploration strategy for a robot with limited view in a grid polygons, having an improved competitive ratio of 5/4. The strategy from Algorithm A requires memory which is linear with respect to the input size. Moreover, by modifying the final step of the algorithm so that the sub-exploration of the cut-off polygon using a recursive call to Algorithm A, rather than by the optimal off-line approach, one can also implement the local actions of the robot in polynomial time, without affecting the competitive ratio of the algorithm.

It would be interesting to ask about the effect of additional restrictions on the memory of the robot on the competitive ratio of the approach. In the wider context of graph exploration, explorations with bounded memory have been the topic of intensive study [4,6,15]. For our problem, it is easy to see that $\Theta(\log C)$ memory is necessary and sufficient to cover the polygon and to terminate at the starting location after $O(C)$ steps, where C is the area of the polygon. This is because the location of the robot can be identified by using coordinates of size $\Theta(\log C)$, whereas exploration of a single column of the polygon, as well as traversal along its boundary, require at most constant memory. It would be interesting to study the trade-off between the amount of allowed memory and the precise value of the competitive ratio of the strategy. One may also ask about competitive strategies in which the robot has extremely limited memory, but is allowed to leave and detect pebbles on the cell of its location and the cells adjacent to it.

Finally, we note that the lower bound on the competitive ratio shown in Section 3 holds for any exploration algorithm in the considered scenario, but it only describes the worst-case performance of algorithms. It is possible that there exist randomized algorithms with a competitive ratio better than 20/17 in expectation; we leave this as a topic for future study.

Acknowledgement

The authors acknowledge the use of resources of TASK Academic Supercomputer Center to run a benchmark of Algorithm A against a test set of 10^{11} small polygons, and thank Jacek Dąbrowski for his kind assistance.

References

1. Arkin, E.M., Fekete, S.P., Islam, K., Meijer, H., Mitchell, J.S.B., Nunez-Rodriguez, Y., Polishchuk, V., Rappaport, D., Xiao, H.: Not being (super)thin or solid is hard: A study of grid hamiltonicity. Computational Geometry: Theory and Applications 42(6-7), 582–605 (2009)

2. Arkin, E.M., Fekete, S.P., Mitchell, J.S.B.: Approximation algorithms for lawn mowing and milling. Computational Geometry: Theory and Applications 17(1-2), 25–50 (2000)
3. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM 45(5), 753–782 (1998)
4. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theoretical Computer Science 345(2-3), 331–344 (2005)
5. Gabriely, Y., Rimon, E.: Competitive on-line coverage of grid environments by a mobile robot. Computational Geometry: Theory and Applications 24(3), 197–224 (2003)
6. Gąsieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 585–594 (2007)
7. Gordon, V.S., Orlovich, Y.L., Werner, F.: Hamiltonian properties of triangular grid graphs. Discrete Mathematics 308(24), 6166–6188 (2008)
8. Grigni, M., Koutsoupias, E., Papadimitriou, C.: An approximation scheme for planar graph TSP. In: Proceedings of the Thirty-Sixth Annual IEEE Symposium on the Foundations of Computer Science (FOCS 1995), pp. 387–411 (1995)
9. Herrmann, D., Kamphans, T., Langetepe, E.: Exploring simple triangular and hexagonal grid polygons online. In: Abstracts of the 24th European Workshop on Computational Geometry, pp. 177–180 (2008)
10. Icking, C., Kamphans, T., Klein, R., Langetepe, E.: Exploring an unknown cellular environment. In: Abstracts of the 16th European Workshop on Computational Geometry, pp. 140–143 (2000)
11. Icking, C., Kamphans, T., Klein, R., Langetepe, E.: Exploring simple grid polygons. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 524–533. Springer, Heidelberg (2005)
12. Itai, A., Papadimitriou, C.H., Szwarcfiter, J.L.: Hamilton paths in grid graphs. SIAM Journal on Computing 11(4), 676–686 (1982)
13. Kamphans, T.: Models and Algorithms for Online Exploration and Search. Ph.D. thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn (2005)
14. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. SIAM Journal on Computing 28(4), 1298–1309 (1999)
15. Reingold, O.: Undirected ST-Connectivity in Log-Space. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005), pp. 376–385 (2005)

An Efficient Self-stabilizing Distance-2 Coloring Algorithm

Jean Blair[1] and Fredrik Manne[2]

[1] Department of EE and CS, United States Military Academy West Point, NY, 10996, USA
Jean.Blair@usma.edu
[2] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
fredrikm@ii.uib.no

Abstract. We present a self-stabilizing algorithm for the distance-2 coloring problem that uses a constant number of variables on each node and that stabilizes in $O(\Delta^2 m)$ moves using at most $\Delta^2 + 1$ colors, where Δ is the maximum degree in the graph and m is the number of edges in the graph. The analysis holds true both for the sequential and the distributed adversarial daemon model. This should be compared with the previous best self-stabilizing algorithm for this problem which stabilizes in $O(nm)$ moves under the sequential adversarial daemon and in $O(n^3 m)$ time steps for the distributed adversarial daemon and which uses $O(\delta_i)$ variables on each node i, where δ_i is the degree of node i.

1 Introduction

The problem of preventing potential interference when assigning frequencies to processes can be modeled as a graph coloring problem where nodes that are sufficiently close must have different colors. As frequencies (colors) are a scarce resource, it is also desirable to use as few colors as possible. A number of different objective functions and models have been studied for this problem; see [1] for a recent survey. In the current paper we study one such problem, that of assigning colors to nodes so that two nodes that are within distance two of each other are assigned different colors. We present and analyse an efficient self-stabilizing algorithm for this problem. The remainder of this section briefly surveys previous work on self-stabilizing coloring algorithms and then shows how the current paper extends and improves on that body of knowledge.

In 1993 Ghosh and Karaata [4] presented an algorithm for coloring planar graphs using at most 6 colors by transforming the graph into a directed acyclic graph, and assuming that all nodes have unique identifiers. This result was later improved to work with bounded variable values and without identifiers by Huang et al. [9] and finally was generalized to a wider class of graphs by Goddard et al. [5].

Also in 1993, Sur and Srimani [14] gave an algorithm for exact coloring of bipartite graphs. The algorithm assumes that a specific node is a root and then colors nodes based on the distance from the root. For this algorithm only finite

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 237–251, 2010.

stabilization was shown and there was no bound on the number of moves. This work was later extended by Kosowski and Kuszner [10] who presented a self-stabilizing algorithm that colors bipartite graphs using exactly two colors and using a polynomial number of moves. Their algorithm also relies on a distinguished root.

Shukla et al. [11] offered randomized self-stabilizing algorithms for coloring of anonymous chains and oriented rings. In [12] the same authors developed self-stabilizing algorithms for two-coloring several classes of bipartite graphs, namely complete odd-degree bipartite graphs and tree graphs.

The first self-stabilizing coloring algorithms for general graphs were given by Gradinariu and Tixeuil [7] in 2000. They presented three different algorithms based on a greedy assignment technique. These algorithms use at most $\Delta + 1$ colors and stabilize in $O(n\Delta)$ moves, where Δ is the maximum node degree in the graph. It is assumed that each node has knowledge of Δ. This result was later improved by Hedetniemi et al. [8] who gave two algorithms for coloring arbitrary graphs, respectively, also using $\Delta + 1$ colors. The moves complexity of these algorithms is $O(n)$ and $O(m)$, where the latter algorithm also guarantees that each node is assigned the smallest available color within its neighborhood.

Other types of coloring problems have also been studied using the self-stabilizing paradigm. For instance, [13] gives a self-stabilizing algorithm that tries to achieve a node coloring where the sum of the colors assigned to each node is minimum. [15] presents a self-stabilizing $\Delta + 4$ edge coloring algorithm for planar graphs in anonymous networks, while [2] describes a self-stabilizing algorithm for edge coloring general graphs.

In this paper we consider self-stabilizing algorithms for the distance-2 coloring problem. That is, one wants to assign colors to the nodes in such a way that each node receives a color different from its neighbors within distance 2 (i.e. different from all of the nodes neighbors and its neighbors' neighbors).

In [6] Gradinariu and Johnen describe a self-stabilizing algorithm for the problem of *unique naming*. This is essentially the same problem as is studied here in that it asks for an assignment of labels to nodes such that no two nodes who are distance-2 neighbors have the same label. They present a randomized scheme where the expected number of moves by each node is one. However, the scheme requires that every node knows n, the number of nodes in the network, and it assigns colors in the range $[1, 2n^2]$.

In [3] Gairing et al. introduce a general mechanism for allowing a node to obtain information at distance-2 from it. The idea is based on each node copying the states of its neighbors and thus making this information available to its own neighbors. It is shown how a distance-2 coloring can be obtained in $O(nm)$ moves under the sequential daemon model and in $O(n^3m)$ time steps under the distributed daemon model. In these algorithms the color of each node can easily be chosen in the range $[1, \delta 2_i + 1]$ where $\delta 2_i$ denotes the number of distance-2 neighbors of node i. We note that the algorithm requires that each node i maintain $O(\delta_i)$ variables where δ_i is the number of neighbors of i.

In the current paper we present a self-stabilizing algorithm for the distance-2 coloring problem that uses at most $\Delta^2 + 1$ colors. The algorithm stabilizes in $O(\Delta^2 m)$ moves under the sequential daemon and also uses the same number of time steps for the distributed daemon model. For a fair daemon (sequential or distributed) our algorithm requires $O(\Delta m)$ rounds to stabilize. In addition, each node is only required to maintain a constant number of variables. Thus our algorithm improves the time step complexity for the distributed adversarial daemon by at least a factor of n and depending on how Δ^2 compares with n the algorithm might also improve the moves complexity for the sequential adversarial daemon. For instance, for a graph where the degree of each node is at most a constant, our algorithm improves the moves complexity by a factor of n for the sequential adversarial daemon and by a factor of n^3 for the distributed adversarial daemon. Moreover, our algorithm improves the overall memory consumption from $O(m)$ down to $O(n)$ variables.

The rest of this paper is organized as follows. In Section 2 we give a short introduction to the self-stabilizing model. In Section 3 we present and motivate our algorithm. In Section 4 we show that any stable configuration of the algorithm also gives a valid distance-2 coloring and in Section 5 we analyze the complexity of the algorithm. Finally, we conclude in Section 6.

2 Model

A system consists of a set of processes where two adjacent processes can communicate with each other. The communication relation is typically represented by a graph $G = (V, E)$ where $|V| = n$ and $|E| = m$. Each process corresponds to a node in V and two nodes i and j are adjacent if and only if $(i, j) \in E$. We assume that each node has a unique identifier. In the following we will not distinguish between a node and its identifier.

The set of neighbors of a node $i \in V$ is denoted by $N(i)$ and $N[i] = N(i) \cup \{i\}$. Similarly we define $N^2(i)$ as the set of neighbors of node i within distance 2 of i and $N^2[i] = N^2(i) \cup \{i\}$. Let $\delta_i = |N(i)|$ and $\Delta = \max_{i \in V} \delta_i$.

A node maintains a set of local variables which make up the local state of the node. Each variable ranges over a fixed domain of values. Every node executes the same algorithm, which consists of one or more rules. A rule has the form name : **if** *guard* **then** *command*. A guard is a boolean predicate over the variables of both the node and those of its neighbors. A command is a sequence of statements assigning new values to the variables of the node.

An assignment of a value to every variable of each node from its corresponding domain defines a configuration of the system. A rule is enabled in some configuration if the guard is true with the current assignment of values to variables. A node is eligible if it has at least one enabled rule. A computation is a maximal sequence of configurations such that for each configuration s_i, the next configuration s_{i+1} is obtained by executing the command of at least one rule that is enabled in s_i. (A node that executes such a rule makes a move or a step). A configuration is defined as stable if there are no eligible nodes in the system.

A daemon is a predicate on executions. We distinguish several kinds of daemons: the sequential daemon makes the system move from one configuration to the next by executing exactly one enabled rule, while the distributed daemon achieves this by executing any non-empty subset of enabled rules. Note that a sequential daemon is an instance of the distributed daemon. Also, a daemon is fair if any rule that is continuously enabled is eventually executed, and adversarial if it may execute any enabled rule at every step. Again, the adversarial daemon is more general than the fair daemon.

A system is self-stabilizing for a given specification if in finite time it converges to a stable configuration that conforms to this specification, independent of its initial configuration and without external intervention.

We consider two measures for evaluating complexity of self-stabilizing programs. A step is the minimum unit of time such that a process can perform any of its moves. For a sequential daemon exactly one process executes one eligible rule during each step, while for a distributed daemon there can be several processes that each makes one simultaneous move during a given step. Thus, the *step complexity* measures the maximum number of steps that are needed to reach a configuration that conforms to the specification (i.e. a legitimate configuration) for all possible starting configurations. The *round complexity* considers that executions are observed in rounds: a round is the smallest sub-sequence of an execution in which every process that was eligible at the beginning of the round either makes a move or has its guard(s) disabled since the beginning of the round. Note that both of these types of analysis focus on communication and not on computation, as it is assumed that a process can perform any type of necessary local computation during one move.

3 The Algorithm

In the following we motivate and describe the new algorithm. We begin by comparing the algorithm with previous self-stabilizing coloring algorithms. In doing so, we examine how coloring conflicts at distance-1 and distance-2 are handled.

For coloring conflicts between neighboring nodes any self-stabilizing algorithm must avoid the possibility of two adjacent nodes repeatedly changing their colors to the same color in a lockstep fashion. With a sequential daemon this is straight forward to handle [8]. For a distributed daemon one can solve this by using a randomized scheme if the network is anonymous [7], or if the nodes have unique identifiers by using the relative values of the identifiers to break ties [7].

For coloring conflicts between nodes at distance-2 there are two issues to consider: how to discover a coloring conflict and then how to resolve it. Even for a sequential daemon, resolving a conflict can be difficult, as information does not propagate immediately between distance-2 neighbors.

Gairing et al. [3] let each node maintain a local copy of the colors of its neighbors. Thus a node i has direct access to the colors of the nodes in $N^2[i]$ and can itself discover any coloring conflicts that it is involved in. A node that wants to change its color must then obtain permission from all of its distance

one neighbors before doing so. This is achieved by using pointers. In this way no two nodes at distance two from each other can change color at the same time.

In the algorithm by Gradinariu and Johnen [6] coloring conflicts are detected by a node i that discovers that it has two neighbors with the same color (one "neighbor" may in fact be i itself). The node i then sets a flag value equal to the conflicting color. This signals that any node in $N[i]$ using this color should recolor itself. Nodes that are affected by this then choose a new color randomly from a predetermined interval.

In our algorithm we combine ideas from both [3] and [6]. A coloring conflict is detected by any node that is adjacent to the conflicting nodes. This node will then signal to exactly one of the conflicting nodes i that it should change its color. When i sees the signal it will put up a flag requesting to change its color. However, i can only recolor itself once all of its neighbors have acknowledged the flag by themselves pointing to i. In this way no other node in $N^2[i]$ can change its color at the same time as i. To select the appropriate color we use a novel deterministic scheme where i will perform a linear search starting from color 1 until it finds a valid color. Each possible color that i considers must either be accepted or rejected by the neighbors of i. If any neighbor rejects the suggested color, i will try the next possible color and repeat until it finds a color that is accepted by all of its neighbors.

A recoloring can either take place because of a distance-1 or a distance-2 coloring conflict. In addition we also force recoloring if the color of a node is higher than a reasonable upper bound on the size of its distance-2 neighborhood. This assures that the final coloring never uses more than $\Delta^2 + 1$ colors.

The following list gives the variables that are available on each node i.

- $dist1deg_i$, the size of $|N[i]|$.
- $dist2deg_i$, an upper bound on the number of nodes in $N^2[i]$. Every node should get a color in the range $[1, dist2deg_i]$.
- c_i, the color of node i.
- $flag_i$, true if node i wants to change its color, otherwise false.
- p_i, a pointer to a node $j \in N[i]$, signalling that j should change its color. If no such node exists then $p_i = null$.
- s_i, the current color of p_i.
- t_i, a color that p_i could change to.
- $coloring_i$, true if node i is in the process of recoloring itself. This requires that $p_j = i$ for all $j \in N[i]$.

Next, we describe two functions that are used by the algorithm. Here node i is the calling processor and in the *NextColor* function $q \in N[i]$.

NextColor(i, q) is used by node i for calculating which color node q could have. The function returns both the current color of q and the smallest color $\geq c_q$, that q can have without causing any coloring conflicts with nodes in $N[i] - \{q\}$.

NextColor(i, q)**:**
$w = \min\{a : a \geq c_q \wedge (\forall z \in N[i] - \{q\} : a \neq c_z)\}$
return (c_q, w)

CorrectPointer(i) is used for determining the next node in $N[i]$ that should change its color (or at least have it verified). A node $j \in N[i]$ needs to attempt a recoloring if either $flag_j = true$ or if $\exists k \in N[i] - \{j\}$ such that $c_j = c_k$. If there are several candidates the one with the lowest ID is chosen. The function returns a triplet (q, c_q, w) where q is the next node in $N[i]$ that should attempt a recoloring and w is the smallest color $\geq c_q$ that does not cause a conflict with nodes in $N[i] - \{q\}$.

CorrectPointer(i):
$q = \min\{\, j \in N[i] : (flag_j = true \vee \exists k \in N[i] - \{j\} : (c_j = c_k))\}$
if $q \neq null$
then return (q, **NextColor**(i, q))
return $(null, null, null)$

Before formally specifying the algorithm, we give the intuition for each rule.

Distance-1: Set $dist1deg_i$ to the size of $N[i]$.

Distance-2: Set $dist2deg_i$ to an upper bound on the size of $N^2[i]$. Note that this rule double counts two nodes in $N(i)$ if they are themselves neighbors or if they have a common neighbor.

Reset: Set $coloring_i$ to $false$ if it is incorrectly $true$. It could be that either $coloring_i$ was incorrectly $true$ in an initial configuration or that i has to abandon an attempt to recolor itself. This is detected if some node $j \in N[i]$ does not point to i (i.e. $p_j \neq i$) or if $flag_i \neq true$.

Notify neighbor: Set p_i to point to the lowest numbered node $j \in N[i]$ that either wants to recolor itself (i.e. $flag_j = true$) or needs to recolor itself because it has a color that conflicts with some node in $N[i]$. Also, set t_i to a suggested new color for $p_i = j$ and set $s_i = c_j$ to indicate that the values have been set in response to the current value of c_j. Note that once a node j has started to recolor itself, as indicated by $coloring_j = true$, no node i that is pointing to j can change its pointer-value. That is, p_i must continue to point to j as long as j is recoloring itself.

Respond to color: If the neighbor p_i is recoloring itself and has changed its color, acknowledge the color change in s_i and if the color s_i conflicts with a color in $N[i]$, use t_i to suggest the next higher possible color for p_i to use. Recall that if the node p_i is recoloring itself (indicated by $coloring_{p_i} = true$) then the node p_i will cycle through possible colors. For each such color, node i must acknowledge the color change (by setting s_i to the new color) and signalling if it accepts the new color (by setting $t_i = c_{p_j}$) or if p_i should change to a higher color ($t_i > c_{p_j}$).

Need new color: If i needs to recolor, set $flag_i = true$, signalling a request to recolor. If a node $j \in N[i]$ is pointing to i ($p_j = i$) while both acknowledging the current color of i ($s_j = c_i$) and requesting that i change its color ($t_j > c_i$), then node i must perform a recoloring. Node i signals to its neighbors that it wants to do so by setting $flag_i = true$. Alternatively, if i has $dist2deg_i < c_i$ then it should also set $flag_i = true$ to indicate that it needs to change its color. Note that the only place that i can later set $flag_i = false$ is in the *Done recoloring* method.

Start recoloring: If every node in $N[i]$ agrees that i is the next to recolor, i begins the recoloring process by setting $coloring_i = true$ and starting with color 1. A node can only start to recolor itself when it has set $flag_i = true$ and each node $j \in N[i]$ is pointing to it ($p_j = i$), while at the same time acknowledging the current color of i (by setting $s_j = c_i$). The node i then sets $coloring_i = true$, locking all other nodes in $N[i]$ from changing their p-values until i has completed the recoloring.

Change color: If all neighbors have acknowledged the current color c_i and at least one neighbor knows of a conflict with c_i, then change i's color. Whenever node i has proposed a new color it must wait for this to be acknowledged by all nodes in $N[i]$ ($s_j = c_i$). If at least one $j \in N[i]$ indicates that there is a conflict with the current color choice (by setting $t_j > c_i$) then i must try the next possible color (i.e., the maximum color over all t_j values).

Done recoloring: If all neighbors have acknowledged the current color c_i and no neighbor knows of a conflict with c_i, set $flag_i = false$ and $coloring_i = false$, indicating that i has completed its recoloring process. Note that in this case there is no distance-2 conflict with c_i. Note also that this is the only routine that sets $flag_i$ to false.

The rules are executed in the given order, meaning that a rule is never executed unless all the previous rules cannot be executed.

Algorithm 1

Distance-1:
if $dist1deg_i \neq |N[i]|$
then $dist1deg_i = |N[i]|$

Distance-2:
if $dist2deg_i \neq (\sum_{j \in N(i)} dist1deg_j) - dist1deg_i + 2$
then $dist2deg_i = (\sum_{j \in N(i)} dist1deg_j) - dist1deg_i + 2$

Reset:
if $(coloring_i = true)$ **and** $((\exists j \in N[i] : p_j \neq i)$ **or** $flag_i = false)$
then $coloring_i = false$

Notify neighbor:
if $(p_i = null$ **or** $coloring_{p_i} = false)$ **and** $((p_i, s_i, t_i) \neq$ **CorrectPointer**$(i))$
then $(p_i, s_i, t_i) =$ **CorrectPointer**(i)

Respond to color:
if $(p_i \neq null)$ **and** $(coloring_{p_i} = true)$ **and** $((s_i, t_i) \neq$ **NextColor**$(i, p_i))$
then $(s_i, t_i) =$ **NextColor**(i, p_i)

Need new color:
if $(flag_i = false)$ **and** $((\exists j \in N[i] : (p_j = i \wedge s_j = c_i \wedge t_j > c_i)) \ \vee$
$(1 \leq dist2deg_i < c_i))$
then $flag_i = true$

Start recoloring:
if $(flag_i = true)$ **and** $(\forall j \in N[i] : (p_j = i \wedge s_j = c_i))$ **and** $(coloring_i = false)$
then $coloring_i = true$
$c_i = 1$

Change color:
if $(coloring_i = true)$ **and** $(\forall j \in N[i] : (p_j = i \wedge s_j = c_i))$ **and** $(\exists j \in N[i] : t_j > c_i)$
then $c_i = max\{t_j : j \in N[i]\}$

Done recoloring:
if $(coloring_i = true)$ **and** $(\forall j \in N[i] : (p_j = i \wedge s_j = c_i \wedge t_j = c_i))$
then $coloring_i = false$
$flag_i = false$

Figure 1 shows a possible execution of Algorithm 1. The initial graph consists of four nodes i, j, k, and l where $i > k$ and with colors as shown in Figure 1a. We assume that the *Distance-1*, *Distance-2*, and *Reset* rules have stabilized before our example starts. Since $c_i = c_k$ node j will first execute a *Notify neighbor* move and set $p_j = k$, $s_j = 2$, and $t_j = 3$. This will force node k to execute a *Need new color* move and set $flag_k = true$. This will again be followed by nodes k and l executing *Notify neighbor* moves giving the configuration shown in Figure 1b. At this point all nodes in $N[k]$ are pointing to k, each with an s-value equal to c_k. Since $t_j > c_k$ it follows that k now can execute a *Start recoloring* move, setting $coloring_k = true$ and $c_k = 1$. From this point no node in $N[k]$ can change its p-value until $coloring_k = false$.

All three nodes in $N[k]$ are now ready to respond to the current value of c_k through *Respond to color* moves. In doing so both nodes j and k will set their t-values $> c_k$ since both of them can see that $c_j = c_k$. This will give the configuration in Figure 1c.

Now k will execute two *Change color* moves, each followed by all nodes in $N[k]$ acknowledging the change in color by executing a *Respond to color* move. This will first increase the value of c_k to 3 (Figure 1d) and then to 4 (Figure 1e). At this point there are no conflicts between c_k and the nodes in $N^2(i)$. This

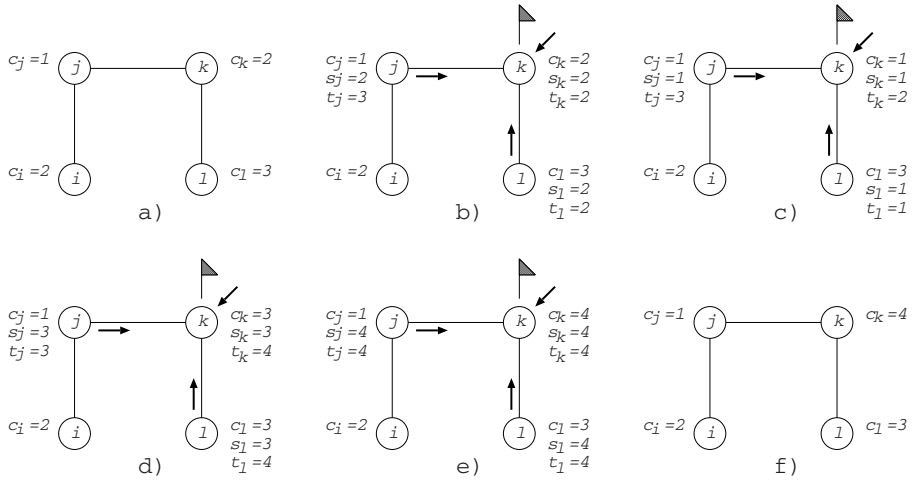


Fig. 1. A possible execution of Algorithm 1

is indicated by the fact that all t-values in $N[k]$ are equal to c_k. Thus node k can execute a *Done recoloring* move which will again be followed by each node in $N[k]$ executing a *Notify neighbor* move to set their p, s, and t values to *null*, finally giving the coloring shown in Figure 1f.

4 Correct Stabilization

In this section we show that when Algorithm 1 is stable the c_i values define a legal distance-2 coloring where no node has a color that is larger than $\delta_i\Delta+1 \leq \Delta^2+1$. We start by showing that each node has an effective bound on the size of $N^2[i]$.

Lemma 1. *In a stable configuration every node i has $dist2deg_i \leq \delta_i\Delta + 1$.*

Proof. Note first that in a stable configuration it follows from the *Distance-1* rule that every node must have $dist1deg_i = \delta_i + 1 \leq \Delta + 1$. The *Distance-2* rule then implies that $dist2deg_i = (\Sigma_{j\in N(i)} dist1deg_j) - dist1deg_i + 2 = (\Sigma_{j\in N(i)}(\delta_j + 1)) - \delta_i + 1 \leq \delta_i\Delta + 1$. ■

Since, for any node i, we have that $|N^2[i]| \leq \delta_i\Delta + 1$, it follows that it is possible to achieve a legal distance-2 coloring where i has a color in the range $[1, \delta_i\Delta + 1]$. To see this, it is sufficient to note that there must be a color in the range $[1, |N^2[i]|]$ not used by the nodes in $N^2(i)$. This color can always be assigned to node i.

Next, we show that when the algorithm is stable no node is actively trying to change color.

Lemma 2. *In any stable configuration, $coloring_i = false$ for every node i.*

Proof. If there exists a node i with $coloring_i = true$, then every node $j \in N[i]$ must have $p_j = i$, otherwise i could execute a *Reset coloring* move. Similarly, there must be at least one node $j \in N[i]$ with $s_j \neq c_i$ or $t_j \neq c_i$ (or both); otherwise i could execute a *Done recoloring* move. A node $j \in N[i]$ where $s_j \neq c_i$ is eligible for a *Respond to color* move, since c_i is *NextColor*(j, i)'s first return value. Thus we may assume that some j has $t_j \neq c_i$. If $t_j < c_i$ then again node j is eligible for a *Respond to color* move, while if $t_j > c_i$ then i is eligible for a *Change color* move. This is a contradiction. It follows that $coloring_i = false$ in a stable configuration. ■

Lemma 3. *In any stable configuration the following statements are true for every node i: (i) $flag_i = false$, (ii) For every pair of distinct nodes $j, k \in N[i], c_j \neq c_k$, and (iii) $c_i \leq dist2deg_i$.*

Proof. This proof is omitted due to space limitations.

We can now state the main result of this section.

Theorem 1. *In a stable configuration the c values define a legal distance-2 coloring where every node i satisfies $c_i \leq \delta_i\Delta + 1$.*

Proof. This follows directly from Lemmas 1 and 3. ■

5 Step Complexity

In this section we derive and prove a bound on the number of time steps needed for Algorithm 1 to stabilize, given an arbitrary initial configuration. The analysis assumes a distributed adversarial daemon. This means that in each time step a non-empty subset of eligible nodes makes one move each.

Table 1 is a summary of upper bounds on the number of time steps that might include a move of each type (i.e., each rule) before stabilization. The last column in the table references the result that proves the number of steps. The results and proofs follow the table.

Lemma 4. *There can be at most* $2(m+n)$ *time steps containing* Distance-1 *or* Distance-2 *moves.*

Proof. Each node can at most make one *Distance-1* move. After this move a node can make one initial *Distance-2* move and then only after each node in $j \in N(i)$ changes its $dist1deg_j$ value. Thus a node i can at most make a total of $\delta_i + 2$ *Distance-1* and *Distance-2* moves. Since $\sum_{i \in V}(\delta_i + 2) = 2n + 2m$ we get that the total number of *Distance-1* and *Distance-2* moves is bounded by $2(m+n)$. ■

Lemma 5. *There can be at most* n *time steps containing* Reset *moves that start with* `coloring = true` *and* `flag = false`.

Proof. Each node i can make one such initial *Reset* move. Any subsequent *Reset* move must follow a *Start recoloring* move and come before any *Done recoloring*

Table 1. Summary of Step Complexity

Move	# Steps (Upper Bound)	Complexity	Proof
Distance-1	n	$= O(m)$	Lemma 4
Distance-2	$n + 2m$	$= O(m)$	Lemma 4
Reset `coloring = false`	0	$= O(1)$	Lemma 5
Reset `coloring = true` `flag = false`	n	$= O(m)$	Lemma 5
Reset `coloring = true` `flag = true`	$n + 8m\Delta$	$= O(\Delta m)$	Lemma 9
Notify neighbor	$9n + 16m$	$= O(m)$	Lemma 12
Respond to color	$4n + 14m(\Delta^2 + \Delta + 1)$	$= O(\Delta^2 m)$	Lemma 14
Need new color	$3n$	$= O(m)$	Lemma 11
Start recoloring	$5n + 8m\Delta$	$= O(\Delta m)$	Lemma 10
Change color	$4n + 8m\Delta$	$= O(\Delta m)$	Lemma 13
Done recoloring	$4n$	$= O(m)$	Corollary 2

move, since this is the only occasion when $coloring_i = true$. However, *Start recoloring* is only executed when $flag_i = true$ and the only move that can set $flag_i = false$ is *Done recoloring*, which also sets $coloring_i = false$. Thus any subsequent *Reset* move cannot be triggered by $flag_i = false$. ■

Before investigating the step complexity of the remaining rules, we examine how each move can or cannot cause a transition between different states of a node i. The states we are interested in depend on the possible values of $coloring_i$ and $flag_i$. Figure 2 shows the state transition diagram. Note that four rules are not shown in the figure since they do not impact the analysis: *Distance-1*, *Distance-2*, *Respond to color*, and *Notify neighbor*.

The transitions in Figure 2 are defined by the predicates and commands of the rules. If $coloring_i = true$ while $flag_i = false$ then i will execute a *Reset* move and set $coloring_i = false$. From this configuration the only move that can affect the values of $coloring_i$ and $flag_i$ is a *Need new color* move that sets $flag_i = true$. From that state the only possible move is *Start recoloring*, which sets $coloring_i = true$. From the configuration $coloring_i = true$ and $flag_i = true$ node i can execute a number of *Change color* moves, but these do not change the values of either $coloring_i$ or $flag_i$. It is possible that i executes a *Reset* move, setting $coloring_i = false$, if some $j \in N[i]$ has $p_j \neq i$. The other possibility is that i executes a *Done recoloring* move and sets $coloring_i = false$ and $flag_i = false$. In addition to these moves, i can also execute a *Distance-1*, a *Distance-2*, a *Notify neighbor*, or a *Respond to color* move. These do not affect $coloring_i$ and $flag_i$ and are not shown in Figure 2.

A *recoloring sequence* by node i consists of a sequence of moves beginning with *Start recoloring* (the transition from state D to state C) and ending with *Done recoloring* (the transition from state C to state B). Note that i can abort an initiated recoloring sequence by executing a *Reset* move and transitioning from state C back to state D. This can only happen if some $j \in N(i)$ executes a *Notify neighbor* move, which will then set $p_j \neq i$, during the same step that

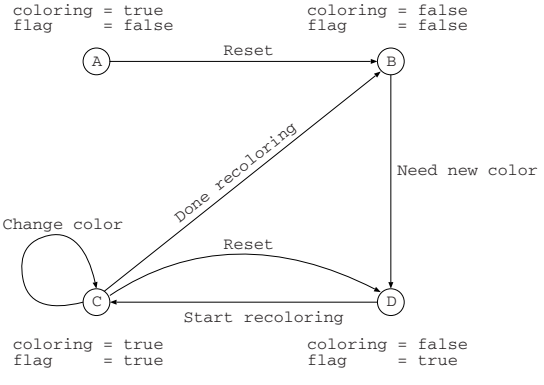


Fig. 2. States of Algorithm 1 with respect to *coloring* and *flag* values

i executes the initial *Start recoloring* move. Otherwise $p_j = i$ will remain true as long as $coloring_i = true$. If i does not abort, we call the recoloring sequence *complete*. A complete recoloring sequence is *correct* if i has been assigned a color $r \leq \delta_i \Delta + 1$ not used by any node in $N^2(i)$ when the recoloring sequence ends.

We now consider a complete recoloring sequence α executed by a node i where α is not the first complete recoloring sequence executed by i. Let g be the time step when i enters α by executing a *Start recoloring* move and let h be the time step when i executes its first *Change color* or *Done recoloring* move in α, whichever comes first. Also, for a particular $j \in N[i]$, let f be the last time step prior to g when j executes a *Notify neighbor* move. Note that j sets $p_j = i$ in time step f. Then $f < g < h$ and $p_j = i$ will remain true for at least the time span $[f, g-1]$ and also after time step $h-1$. In the same manner $coloring_i = true$ in the time span $[g, h]$ (and possibly longer).

Our next result considers the values that t_j can take on prior to time step h.

Lemma 6. *Let g be the time step when node i executes the* Start recoloring *move in a non-initial complete recoloring sequence α, and let h be the earliest time step in α that i executes a* Change color *or* Done recoloring *move. For any particular $j \in N[i]$, let f be the last time step prior to g when j executes a* Notify neighbor *move.*

After time step $h-1$ and before time step h: for every $j \in N[i]$ the following are true: $p_j = i$, $s_j = c_i$, and either $t_j = 1$ or t_j is equal to the lowest or second lowest unused color in $N[j] - \{i\}$.

Proof. This proof is omitted due to space limitations.

Now we have established the different possible values that each t_j for $j \in N[i]$ can have just after time step $h-1$. The next two results are needed to make sure that i starts to select a new color once i has executed a *Start recoloring* move.

Corollary 1. *Let g be the time step when node i executes the* Start recoloring *move in a non-initial complete recoloring sequence α, and let h be the earliest time step in α that i executes a* Change color *or* Done recoloring *move.*

If the Need new color *move by i that set $flag_i = true$ prior to i entering α was caused by $dist2deg_i < c_i$ then for each $j \in N[i]$ the value of t_j will be pointing to the lowest unused color in $N[j] - \{i\}$ after time step $h-1$.*

Proof. The *Need new color* rule requires that $1 \leq dist2deg_i < c_i$. Thus since $1 < c_i$ the value of c_i will be reduced to 1 when i executes a *Start Recoloring* move in time step g. ■

Lemma 7. *Let g be the time step when node i executes the* Start recoloring *move in a non-initial complete recoloring sequence α, and let h be the earliest time step in α that i executes a* Change color *or* Done recoloring *move.*

If the Need new color *move by i that set $flag_i = true$ prior to i entering α was not caused by $dist2deg_i < c_i$ then there must be some $j \in N[i]$ that has $t_j > 1$ after time step $h-1$.*

Proof. Since i exited the previous recoloring sequence with every $j \in N[i]$ satisfying $t_j = c_i$ at the time step in which the *Done recoloring* move was executed, there must exist some node $j \in N[i]$ that executed a *Notify neighbor* move and set $p_j = i$, $s_j = c_i$, and $t_j > c_i$ prior to i executing the *Need new color* move to enter α. At the time j executed the *Notify neighbor* move, there must have existed at least one node $k \in N[j] - \{i\}$ such that $c_k = c_i$ and $i < k$. Note that k cannot have changed color between this point and time step h, because $p_j \neq k$. (As long as $c_i = c_k$ *CorrectPointer(j)* will never set $p_j = k$ in *Notify neighbor* since $i < k$.) Thus we can conclude that the coloring conflict between i and k still exists after time step $h - 1$. From this it follows that when p_j was last set to i the value of t_j must have been set to a value greater than c_i. ■

We can now show that α must be correct.

Lemma 8. *Let α be a recoloring sequence for node i. When i exits α there is no node in $N^2(i)$ with the same color as i and $c_i \leq \delta_i \Delta + 1$.*

Proof. This proof is omitted due to space limitations.

Note that every node $j \in N[i]$ must execute at least one *Respond to color* move before i executes a *Done recoloring* move. Thus the value of $dist1deg_j$ for each $j \in N[i]$ must be correct when i exits α. Similarly, $dist2deg_i$ must be correct when i exits α.

We have now shown that every complete recoloring sequence (except maybe the first) will result in a node i having a distinct color among all the nodes in $N^2[i]$. However, there is a possibility that i does not complete a recoloring sequence and this may result in a coloring conflict. But as the proof of the following result shows, the non-complete recoloring sequences can be subsumed in the complete recoloring sequences.

Theorem 2. *No node will perform more than three complete recoloring sequences.*

Proof. From Lemma 8 it follows that a non-initial complete recoloring sequence by a node i will result in c_i being unique relative to the colors used by the nodes in $N^2(i)$. An incomplete recoloring sequence by i will result in i executing a *Reset* move with $c_i = 1$. Thus if i has received a legal color such that $c_i > 1$, then no node in $N^2(i)$ will receive the same color as i.

Now assume a node i has executed its second complete recoloring sequence. If $c_i > 1$ then no node $k \in N^2(i)$ can exit a subsequent recoloring sequence with $c_k = c_i$. But if k, where $i < k$, performs a *Reset* move right after executing a *Start recoloring* move and if $c_i = 1$ then we get $c_i = c_k$. This would force i to perform a new recoloring sequence which would result in $c_i > 1$ (since k cannot change color until i has done so) and thus no further recoloring sequences would be needed by i. ■

Now that we have shown that each node can execute at most three complete recoloring sequences it is fairly straight forward to count the number of different moves each node can make. The following results state these counts without showing the straight-forward proofs, in the interest of saving space.

Corollary 2. *There can be at most* $4n$ *time steps containing* Done recoloring *moves.*

Lemma 9. *There can be at most* $n + 8m\Delta$ *time steps containing* Reset *moves that start with* `coloring = true` *and* `flag = true`.

Lemma 10. *There can be at most* $5n + 8m\Delta$ *time steps containing* Start recoloring *moves.*

Lemma 11. *There can be at most* $3n$ *time steps containing* Need new color *moves.*

Lemma 12. *There can be at most* $9n + 16m$ *time steps containing* Notify neighbor *moves.*

Lemma 13. *There can be at most* $4n + 8m\Delta$ *time steps containing* Change color *moves.*

Lemma 14. *There can be at most* $4n + 14m(\Delta^2 + \Delta + 1)$ *time steps containing* Respond to color *moves.*

Theorem 3. *Algorithm 1 stabilizes after* $O(\Delta^2 m)$ *time steps.*

Proof. The result follows directly from Lemmas 4, 5, 9, 10, 11, 12, 13, 14 and Corollary 2. See Table 1 for a summary. ■

We note that the same time step analysis holds for a sequential adversarial daemon. The main difference between a distributed and sequential adversarial daemon is that with the sequential one, we can show that any node that has gone through at least two complete recoloring sequences will end up with the lowest color not used by any node in $N^2(i)$, as opposed to the second lowest for the distributed daemon. However, in both cases one cannot guarantee that each node has been assigned the lowest available color in a stable solution, as there might be nodes that do not change color during the execution of the algorithm.

The analysis for a fair daemon (sequential or distributed) is not much different from the one presented here and gives a round complexity of $O(\Delta m)$. Although we omit the details due to space considerations it is not hard to see that when a node i has executed a *Change color* move, all nodes in $N[i]$ can respond to this in one round. Thus the complexity of the *Respond to color* moves, which are the most frequent moves, are lowered from $O(\Delta^2 m)$ moves for the adversarial daemon to $O(\Delta m)$ rounds for the fair daemon. To see that this is also a lower bound it is sufficient to consider a complete graph where every node starts with the same initial color.

6 Concluding Remarks

We note that Algorithm 1 can easily be modified to solve various other restricted coloring problems. For instance, colors could be selected from a finite list of

available colors (a so called *list coloring*) or it could be required that $|c_i - c_j| > a$ when i and j are distance-2 neighbors, where a is some positive constant.

However, Algorithm 1 cannot in its current form produce a Grundy coloring (i.e. where each node i has the lowest available color in $N^2(i)$) as it cannot detect available free colors that are smaller than the current (correct) color. One solution to this could be to let each node set $flag_i = true$ with some small probability.

We also note that although we require that every node has a unique identifier, it is not hard to show that it suffices that each identifier is unique within distance-2 for the algorithm to run correctly.

References

1. Aardal, K.I., van Hoesel, S.P.M., Koster, A.M.C.A., Mannino, C., Sassano, A.: Models and solution techniques for frequency assignment problems. Ann. Op. Res. 153, 79–129 (2007)
2. Chaudhuri, P., Thompson, H.: A self-stabilizing distributed algorithm for edge-coloring general graphs. Aust. J. Comb. 38, 237–248 (2007)
3. Gairing, M., Goddard, W., Hedetniemi, S.T., Kristiansen, P., McRae, A.A.: Distance-two information in self-stabilizing algorithms. Par. Proc. L. 14, 387–398 (2004)
4. Ghosh, S., Karaata, M.H.: A self-stabilizing algorithm for coloring planar graphs. Dist. Comp. 7, 55–59 (1993)
5. Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: Self-stabilizing algorithms for orderings and colorings. Int. J. Found. Comp. Sci. 16, 19–36 (2005)
6. Gradinariu, M., Johnen, C.: Self-stabilizing neighborhood unique naming under unfair scheduler. In: Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L. (eds.) Euro-Par 2001. LNCS, vol. 2150, pp. 458–465. Springer, Heidelberg (2001)
7. Gradinariu, M., Tixeuil, S.: Self-stabilizing vertex coloring of arbitrary graphs. In: OPODIS 2000, pp. 55–70 (2000)
8. Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: Linear time self-stabilizing coloring. Inf. Proc. Lett. 87, 251–255 (2003)
9. Huang, S.-T., Hung, S.-S., Tzeng, C.-H.: Self-stabilizing coloration in anonymous planar networks. Inf. Proc. Lett. 95, 307–312 (2005)
10. Kosowski, A., Kuszner, L.: Self-stabilizing algorithms for graph coloring with improved performance guarantees. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 1150–1159. Springer, Heidelberg (2006)
11. Shukla, S., Rosenkrantz, D., Ravi, S.: Developing self-stabilizing coloring algorithms via systematic randomization. In: Proc. Int. Workshop on Par. Process., pp. 668–673 (1994)
12. Shukla, S.K., Rosenkrantz, D., Ravi, S.S.: Observations on self-stabilizing graph algorithms for anonymous networks. In: Proc. of the Second Workshop on Self-stabilizing Systems, pp. 7.1–7.15 (1995)
13. Sun, H., Effantin, B., Kheddouci, H.: A self-stabilizing algorithm for the minimum color sum of a graph. In: Rao, S., Chatterjee, M., Jayanti, P., Murthy, C.S.R., Saha, S.K. (eds.) ICDCN 2008. LNCS, vol. 4904, pp. 209–214. Springer, Heidelberg (2008)
14. Sur, S., Srimani, P.K.: A self-stabilizing algorithm for coloring bipartite graphs. Inf. Sci. 69, 219–227 (1993)
15. Tzeng, C.-H., Jiang, J.-R., Huang, S.-T.: A self-stabilizing $(\delta + 4)$-edge-coloring algorithm for planar graphs in anonymous uniform systems. Inf. Proc. Lett. 101, 168–173 (2007)

Distributed Computing of Efficient Routing Schemes in Generalized Chordal Graphs*

Nicolas Nisse[1], Ivan Rapaport[2], and Karol Suchan[3,4]

[1] MASCOTTE, INRIA, I3S, CNRS, UNS, Sophia Antipolis, France
[2] DIM, CMM (UMI 2807 CNRS), Universidad de Chile, Santiago, Chile
[3] Facultad de Ingeniería y Ciencias, Univ. Adolfo Ibáñez, Santiago, Chile
[4] WMS, AGH University of Science and Technology, Cracow, Poland

Abstract. Efficient algorithms for computing routing tables should take advantage of the particular properties arising in large scale networks. There are in fact at least two properties that any routing scheme must consider: low (logarithmic) diameter and high clustering coefficient.

High clustering coefficient implies the existence of few large induced cycles. Therefore, we propose a routing scheme that computes short routes in the class of k-chordal graphs, i.e., graphs with no chordless cycles of length more than k. We study the tradeoff between the length of routes and the time complexity for computing them. In the class of k-chordal graphs, our routing scheme achieves an additive stretch of at most $k-1$, i.e., for all pairs of nodes, the length of the route never exceeds their distance plus $k-1$.

In order to compute the routing tables of any n-node graph with diameter D we propose a distributed algorithm which uses $O(\log n)$-bit messages and takes $O(D)$ time. We then propose a slightly modified version of the algorithm for computing routing tables in time $O(\min\{\Delta D, n\})$, where Δ is the the maximum degree of the graph. Using these tables, our routing scheme achieves a better additive stretch of 1 in chordal graphs (notice that chordal graphs are 3-chordal graphs). The routing scheme uses addresses of size $\log n$ bits and local memory of size $2(d-1)\log n$ bits in a node of degree d.

Keywords: Routing scheme, stretch, chordal graph, distributed algorithm.

1 Introduction

In any distributed communication network it is important to deliver messages between pairs of processors. Routing schemes are employed for this purpose. A routing scheme is a distributed algorithm that directs traffic in a network. More precisely, any source node must be able to route messages to any destination node, given the destination's network identifier. When investigating routing

* Partially supported by programs Fondap and Basal-CMM (I.R. and K.S.), Fondecyt 1090156 (I.R.) and the European project IST FET AEOLUS (N.N.).

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 252–265, 2010.

schemes, several complexity measures arise. On one hand, it is desirable to use as short paths as possible for routing messages. The efficiency of a routing scheme is measured in terms of its *multiplicative stretch factor* (resp., *additive stretch factor*), i.e., the maximum ratio (resp., difference) between the length of a route computed by the scheme and that of a shortest path connecting the same pair of nodes. On the other hand, as the amount of storage at each processor is limited, the routing information stored in the processors' local memory, the *routing tables*, must not require too much space with respect to the size of the network. Last but not least, because of the dynamic character of networks, it is important to be able to compute the routing information in an efficient distributed way. While many works propose good tradeoffs between the stretch and the size of routing tables, the algorithms that compute those tables are often impracticable because they are centralized algorithms or because of their time-complexity. Indeed, in the context of large scale networks like social networks or Internet, even polynomial time algorithms are inefficient. In this paper, we focus on the tradeoff between the length of the computed routes and the time complexity of the computation of routing tables.

One way to design efficient algorithms in large scale networks consists in taking advantage of their specific properties. In particular, they are known to have low (logarithmic) diameter and to have high clustering coefficient. Therefore, their chordality (the length of the longest induced cycle) is somehow limited (e.g., see [Fra05]). That is why, in this paper, we focus on the class of k-chordal graphs. A graph G is called *k-chordal* if it does not contain induced cycles longer than k. A 3-chordal graph is simply called *chordal*. This class of graphs received particular interest in the context of compact routing. Dourisboure and Gavoille proposed routing tables of at most $\log^3 n/\log\log n$ bits per node, computable in time $O(m+n\log^2 n)$, that give a routing scheme with additive stretch $2\lfloor k/2\rfloor$ in the class of k-chordal graphs [DG02]. Also, Dourisboure proposed routing tables computable in polynomial time, of at most $\log^2(n)$ bits, but that give an additive stretch $k+1$ [Dou05]. Using a Lexicographic Breadth-First Search (Lex-BFS) ordering (resp., BFS ordering) of the vertices, Dragan designed a $O(n^2)$-time algorithm to approximate the distance up to an additive constant of 1 (resp., up to $k-1$) between all pairs of nodes of any n-node chordal graph (resp., k-chordal graph) [Dra05]. All these time results consider the centralized model of computation.

In this paper we propose a simpler routing scheme which, in particular, can be quickly computed in a distributed way and achieves good additive stretch for k-chordal graphs. However, the simplicity comes at a price of $O(\log n)$ bits per port needed to store the routing tables.

Distributed Model. An interconnection network is modeled by a simple undirected connected n-node graph $G=(V,E)$. In the following, D denotes the diameter of G and Δ denotes its maximum degree. The processors (nodes) are autonomous computing entities with distinct identifiers of size $\log n$ bits. We consider an all-port, full-duplex, $O(\log n)$ bounded message size, synchronous communication model. That is, any processor is able to send (resp., receive)

different messages of size $O(\log n)$ to (resp., from) each of its neighbors in one communication step; links (edges) are bidirectional.

Our results. We present a simple routing scheme using a relabeling of the vertices based on a particular BFS-tree. Using a Strong BFS-tree, our algorithm achieves an additive stretch $k-1$ in the class of k-chordal graphs, and using a Maximum Neighborhood BFS-tree (Max-BFS-tree), it achieves an additive stretch 1 in the class of chordal graphs. It uses addresses of size $\log n$ bits and local memory of size $2(d-1)\log n$ bits per node of degree d. More precisely, each node must store an interval ($2\log n$ bits) per port, except for one port.

The stretches we achieve equal the best ones obtained in previous works. But our algorithm is a (simple) distributed one. It uses messages of size $O(\log n)$ bits. It computes a relabeling of the vertices and the routing tables in time $O(D)$ when a Strong BFS-tree is used, and in time $O(\min\{\Delta D, n\})$ when a Max-BFS-tree is used.

In the class of chordal graphs, our results simplify those of Dragan since a Lex-BFS ordering is more constrained than a Max-BFS ordering. In particular, the design of a distributed algorithm that computes a Lex-BFS ordering of the vertices of any n-node graph G in time $o(n)$ is an open problem even if G has small diameter and maximum degree.

Related work. Two kinds of routing schemes have been studied. In the *name-independent* model, the designer of the routing scheme has no control over the node names (see, e.g., [PU89, GP96, GG01]). Here we focus on *labeled routing*, where the designer of the routing scheme is free to name the nodes with labels containing some information about the topology of the network, the location of the nodes in the network, etc. In this context, a routing scheme with multiplicative stretch $4k-5$, $k \geq 2$, and using $\tilde{O}(n^{1/k})$ bits per node[1] in arbitrary graphs is designed in [TZ01]. In the case of trees, optimal labeled routing schemes using $\tilde{O}(1)$ bits per node have been proposed in [FG01, TZ01]. In [FG01], it is shown that any optimal routing scheme using addresses of $\log n$ bits requires $\Omega(\sqrt{n})$ bits of local memory-space. Several network classes have been studied, like planar graphs [Tho04], graphs with bounded doubling dimension [AGGM06], graphs excluding a minor [AG06], etc.

A particular labeled routing scheme is *interval routing*. Defined in [SK85], interval routing has received particular interest [Gav00]. In such a scheme, the nodes of the network are labeled using integers, and outgoing arcs in a node are labeled with a set of intervals. The set of all the intervals associated to all the outgoing edges of a node forms a partition of the name range. The routing scheme consists in sending the message through the unique outgoing arc labeled by an interval containing the destination's label. The complexity measure is the maximum number of intervals used in the label of an outgoing arc. An asymptotically tight complexity of $n/4$ intervals per arc in an n-node network is given in [GP99]. Moreover, almost all networks support an optimal interval

[1] The notation $\tilde{O}()$ indicates complexity similar to $O()$ up to polylogarithmic factors.

routing scheme using at most 2 intervals per outgoing link [GP01]. Specific graph classes have been studied in this context (e.g., k-trees [NN98]).

2 Generalities on BFS-Orderings and BFS-Trees

In the following, $G = (V, E)$ denotes a connected n-node graph. Let $H = (V(H), E(H))$ be a subgraph of G, i.e., $V(H) \subseteq V$ and $E(H) \subseteq \{\{u, v\} \in E \mid u, v \in V(H)\}$. $d_H(x, y)$ denotes the distance in H between $x, y \in V(H)$. $N_H(x)$ denotes the neighborhood of $x \in V(H)$ in H. The length $|P|$ of a path P is its number of edges. A vertex $v \in V$ is *simplicial* if its neighborhood induces a clique. An ordering $\{v_1, \cdots, v_n\}$ on the vertices of G is called a *perfect elimination ordering* (PEO) if, for any $1 \leq i \leq n$, v_i is simplicial in G_i, where G_i is the graph induced by $\{v_i, \cdots, v_n\}$. In the context of a vertex ordering, we denote $w < v$ if w has a smaller index in this ordering. Note that in a PEO, if $z < w < v$, $\{z, w\} \in E$ and $\{z, v\} \in E$, then $\{w, v\} \in E$.

Theorem 1. *[FG65] A graph is chordal iff it admits a PEO.*

Let $r \in V$. A *Breadth-First Search* (BFS) ordering of G rooted at r is an ordering of its vertices such that r is the greatest vertex and, for any $u, v \in V(G) \setminus \{r\}$, $v < u$ implies that the greatest neighbor of u is greater than or equal to the greatest neighbor of v. A *Maximal Neighborhood Breadth-First Search* (MaxBFS) ordering of G rooted at r is a BFS ordering of its vertices with the following additional constraint: for any $u, v \in V(G) \setminus \{r\}$ with the same greatest neighbor, $v < u$ implies that the number of neighbors of u greater than u is at least the number of neighbors of v greater than u. The following theorem will be widely used.

Theorem 2. *[BKS05, CK] A graph G is chordal if and only if any MaxBFS ordering is a PEO.*

Given an ordering $\mathcal{O}$ of the vertices of G, the spanning tree *defined* by $\mathcal{O}$ is the spanning tree obtained by choosing for each vertex, but the root, its greatest neighbor as the parent. Such a tree defined by a BFS ordering (resp., by a MaxBFS ordering) will be called a Strong BFS-tree[2] (resp., MaxBFS-tree). Such a tree is rooted at the greatest vertex in the ordering.

3 Routing Scheme Using Strong BFS and MaxBFS

This section is devoted to presenting a simple routing scheme based on Strong BFS-trees. We prove that this scheme achieves a good additive stretch in k-chordal graphs, and an improvement of this routing scheme is provided for chordal graphs.

[2] The name BFS-tree is often used in distributed computing literature to denote any shortest paths tree. To emphasize the particular properties of BFS-trees that are used in this work, the authors decided to add "Strong" in the name, even though the BFS-trees found in many textbooks are Strong BFS-trees in this sense.

First, let us present some notation. Let T be a spanning tree of a graph G. Given $x, y \in V$, $T_{x \to y}$ denotes the path in T between x and y. When T is defined by some BFS ordering, for any $v, w \in V$, $v > w$ denotes that v has a bigger index than w in this ordering. When T is rooted at $r \in V(T)$, its vertices are partitioned into *layers*: the layer $\ell(v)$ of a vertex v corresponds to $d_G(v, r)$. Note that $\{u, v\} \in E \Rightarrow |\ell(u) - \ell(v)| \leq 1$. In this paper we consider rooted trees, so we may say that two vertices are in the same *branch* if their least common ancestor is equal to one of them. Finally, given a routing scheme $\mathcal{R}$ on G, $Str(\mathcal{R}, xy)$ denotes the difference between the length of the path computed by $\mathcal{R}$ and the distance in G between x and y. The (additive) stretch $Str(\mathcal{R})$ of $\mathcal{R}$ in G corresponds to $\max_{x,y \in V} Str(\mathcal{R}, xy)$.

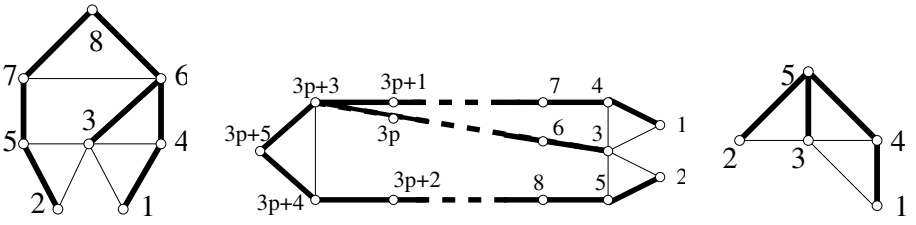
3.1 General Routing Scheme

Let G be a graph and T be any Strong BFS-tree of G. Roughly, the routing scheme we propose proceeds as follows to send a message from any source $x \in V(G)$ to any destination $y \in V(G)$. The message follows the path from x to y in T, but if at some step the message can go through an edge $e \in E(G) \setminus E(T)$ that leads to the branch of T containing y, then it will use this *shortcut*. More formally, our routing scheme $\mathcal{R}(G, T)$ is defined as follows.

If $x = y$, stop.
If there exists $w \in N_G(x)$ ancestor of y in T,
  choose such a vertex w minimizing $d_T(w, y)$;
Otherwise, choose the parent of x in T.

For instance, Figure 1 represents 3 graphs where the spanning trees are depicted with bold edges. In Figure 1(a), a message from 1 to 2 will follow the path $\{1, 4, 6, 7, 5, 2\}$. In Figure 1(c), the same message will follow $\{1, 4, 5, 2\}$. Let us make some simple remarks.

1. The routing scheme $\mathcal{R}(G, T)$ is well defined. Indeed, the message will eventually reach its destination since its distance to y in T is strictly decreasing at each step. Note that even if T is an arbitrary rooted spanning tree of G, the distance from y in T may increase at most once, if the message is passed from a descendant of y to an ancestor at a larger distance from y.
2. Once a spanning tree T rooted at an $r \in V(G)$ has been defined, this scheme can be efficiently implemented. It is sufficient to label the vertices such that any rooted subtree of T corresponds to a single interval. For any $u \in V(G)$ and any neighbor v of u but its parent, u stores the interval corresponding to the subtree of T rooted at v. Then, the routing function chooses the port corresponding to the inclusion-minimal interval containing the destination's address, and it chooses the parent of the current location if no such interval exists. Note that this is not a standard interval routing scheme as defined in [SK85] because some intervals may be contained in others.



(a) 4-chordal graph and Strong BFS-tree. (b) $2p+2$-chordal graph and MaxBFS-tree. (c) Chordal graph and MaxBFS-tree.

Fig. 1. Lemmata 3 and 4 give optimal bounds

3. Since we assume that T is a BFS-tree, it is easy to see that the route computed by the routing scheme $\mathcal{R}(G,T)$ between two arbitrary nodes contains at most one edge that is not an edge of T. Indeed, after having taken such a *shortcut*, the message reaches y by following the path in T, which is a shortest path in G since T is a BFS-tree.

This section is devoted to proving the following theorem.

Theorem 3. *Let $k \geq 3$ and let G be a k-chordal graph.*

- *Let T be* any *Strong BFS-tree of G. Then $Str(\mathcal{R}(G,T)) \leq k-1$.*
- *Let $k=3$ and let T be* any *MaxBFS-tree of G. Then $Str(\mathcal{R}(G,T)) \leq 1$.*
- *Both bounds are tight.*

3.2 Stretch in k-Chordal Graphs

Let $k \geq 3$ and let G be a k-chordal graph and T be a (rooted) Strong BFS-tree of G. Let $x, y \in V$ be an arbitrary source and destination, respectively. The proof is a case by case analysis to bound $Str(\mathcal{R}(G,T), xy)$. Let R_{xy} be the route from x to y computed by $\mathcal{R}(G,T)$. In the following, we compare the length of R_{xy} with the length of some shortest path between x and y in G. Several parts of the following discussion are depicted in Figure 2, where bold lines represent edges, thin lines represent paths belonging to T and dotted lines represent paths with edges not necessarily belonging to T.

Restriction w.l.o.g. In this subsection, we prove that it is sufficient to consider x and y with a smallest common ancestor r_0 such that there is a shortest path P between x and y in G with no internal vertices of P in $V(T_{r_0 \to y}) \cup V(T_{x \to r_0})$.

If x is an ancestor or a descendant of y, R_{xy} is the path between x and y in T. Since T is a BFS-tree, this is a shortest path. From now on, we assume that x and y have a least common ancestor $r_0 \in V(G)$ distinct from x and y. By definition of $\mathcal{R}(G,T)$, R_{xy} either passes through r_0, or it uses an edge $\{e,f\} \in E(G) \setminus E(T)$ with $e \in V(T_{x \to r_0}) \setminus \{r_0\}$ and $f \in V(T_{r_0 \to y}) \setminus \{r_0\}$. I.e., the route R_{xy} from x to y is either $T_{x \to e} \cup \{e,f\} \cup T_{f \to y}$, or $T_{x \to r_0} \cup T_{r_0 \to y}$.

First, we need a technical lemma that shows that the roles of x and y are somehow symmetric.

Lemma 1. $Str(\mathcal{R}(G,T),xy) = Str(\mathcal{R}(G,T),yx)$.

Proof. Let R_{yx} be the route computed by $\mathcal{R}(G,T)$ from y to x. If R_{xy} passes through r_0, then R_{yx} does so, and $R_{xy} = R_{yx}$. If $R_{xy} \neq R_{yx}$, then R_{yx} must contain an edge $\{e',f'\} \in E(G) \setminus E(T)$ other than $\{e,f\}$ (see Fig. 2(a)), and $e' \in V(T_{e\to r_0}) \setminus \{e\}$ and $f' \in V(T_{r_0\to f}) \setminus \{f\}$. Because T is a Strong BFS-tree, e' is the parent of e and f is the parent of f'. Indeed, $d_G(r_0,f) < d_G(r_0,f') \leq d_G(r_0,e')+1 \leq d_G(r_0,e) \leq d_G(r_0,f)+1$. To conclude, if $R_{xy} \neq R_{yx}$, $R_{xy} = T_{x\to e} \cup \{e,f\} \cup \{f,f'\} \cup T_{f'\to y}$, and $R_{yx} = T_{y\to f'} \cup \{f',e'\} \cup \{e',e\} \cup T_{e\to x}$.

Let P_0 be any shortest path in G between x and y. Let y' be the first vertex of P_0 in $V(T_{r_0\to y})$, and x' be the last vertex of P_0, before y', in $V(T_{x\to r_0})$. Let P' be the subpath of P_0 between x' and y'. Because T is a Strong BFS-tree, $P = T_{x\to x'} \cup P' \cup T_{y'\to y}$ is a shortest path between x and y in G. The following technical lemma restricts our investigation to the case when P has no internal vertices in $V(T_{r_0\to y}) \cup V(T_{x\to r_0})$.

Lemma 2. $Str(\mathcal{R}(G,T),xy) = 0$, *or* $Str(\mathcal{R}(G,T),xy) = Str(\mathcal{R}(G,T),x'y')$.

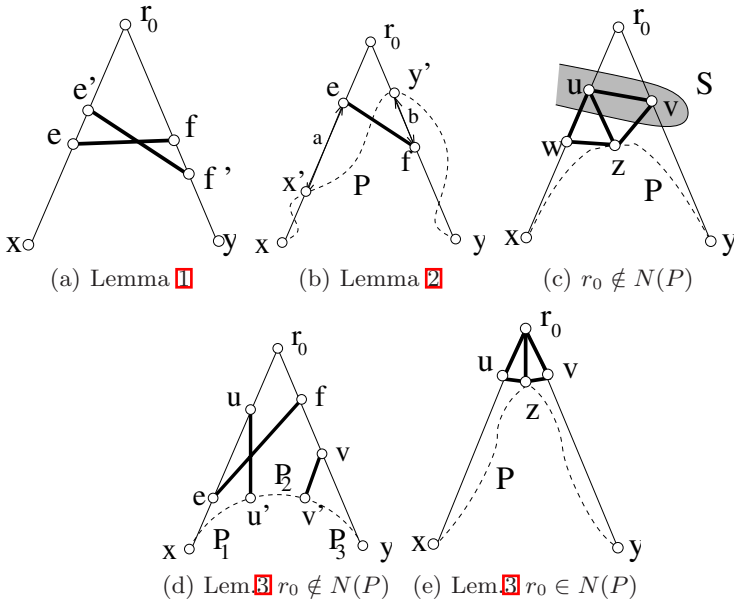
Proof. If $x' = y' = r_0$, then it is easy to see that $|P| = |R_{xy}|$. By definition, x' and y' must be both equal to or different from r_0. Therefore, let us assume both are different from r_0. Recall that $R_{xy} = T_{x\to e} \cup \{e,f\} \cup T_{f\to y}$, or $R_{xy} = T_{x\to r_0} \cup T_{r_0\to y}$. In the second case, we set $e = f = r_0$. The proof is a case analysis according to the relative positions of x' and e in $T_{x\to r_0}$, and of y' and f in $T_{r_0\to y}$.

If $T_{x\to e} \subseteq T_{x\to x'}$ and $T_{f\to y} \subseteq T_{y'\to y}$, then $Str(\mathcal{R}(G,T),xy) = 0$ because $|P'| \geq 1$.

Let us assume that $T_{x\to x'} \subset T_{x\to e}$ and $T_{f\to y} \subset T_{y'\to y}$. This case is illustrated in Figure 2(b). Note that in this case $e \neq f$, therefore $\{e,f\} \in E(G)$. Let $a = |T_{x\to e}| - |T_{x\to x'}| > 0$, and let $b = |T_{y'\to y}| - |T_{f\to y}| > 0$. We study the layers of x, x', y and y' to prove that $Str(\mathcal{R}(G,T),xy) = 0$. Let $L = \ell(e)$ be the layer of e. Then, $\ell(x') = L + a$. Because $\{e,f\} \in E(G)$ and T is a Strong BFS-tree, $L-1 \leq \ell(f) \leq L+1$. Therefore, $L-1-b \leq \ell(y') \leq L+1-b$. However, because T is a Strong BFS-tree and P' is a shortest path between x' and y', we must have $\ell(x') \leq \ell(y') + |P'|$. Thus, $\ell(x') \leq L+1-b+|P'|$. Finally, we get that $a+b-1 \leq |P'|$. Since $b > 0$, then $a \leq |P'|$. To conclude, let us observe that $|R_{xy}| = |T_{x\to x'}| + a + 1 + |T_{y'\to y}| - b = |P| - |P'| + a + 1 - b \leq |P|$. Hence, $Str(\mathcal{R}(G,T),xy) = 0$.

If $T_{y\to y'} \subset T_{f\to y}$ and $T_{e\to x} \subset T_{x'\to x}$, it can be proven in a similar way that $Str(\mathcal{R}(G,T),yx) = 0$. By Lemma 1, we get that $Str(\mathcal{R}(G,T),xy) = 0$.

Finally, if $T_{x\to x'} \subseteq T_{x\to e}$ and $T_{y'\to y} \subseteq T_{f\to y}$, the route computed by $\mathcal{R}(G,T)$ from x' to y' is clearly $T_{x'\to e} \cup \{e,f\} \cup T_{f\to y'}$. Moreover, $Str(\mathcal{R}(G,T),xy) = |T_{x'\to e} \cup \{e,f\} \cup T_{f\to y'}| - |P'| = Str(\mathcal{R}(G,T),x'y')$.



(a) Lemma 1 (b) Lemma 2 (c) $r_0 \notin N(P)$

(d) Lem.3 $r_0 \notin N(P)$ (e) Lem.3 $r_0 \in N(P)$

Fig. 2. Illustrations of the bounds of $Str(\mathcal{R}(G,T),xy)$

Whenever route and shortest path are vertex disjoint It remains to consider the case when x and y have a smallest common ancestor r_0 such that there is an xy-shortest path P in G with no internal vertices of P in $V(T_{r_0\to y}) \cup V(T_{x\to r_0})$ (cf. Figures 2(c), 2(d), 2(e)). Basically, the proof proceeds by considering the distances in the cycle $T_{x\to r_0} \cup T_{y\to r_0} \cup P$ and finding convenient chords in it.

Claim. If $r_0 \notin N_G(P)$, there exist u in $T_{x\to r_0}$, and v in $T_{y\to r_0}$, such that $u, v \in N_G(P)$. Moreover, if G is chordal, u and v may be chosen adjacent: $\{u,v\} \in E(G) \setminus E(T)$.

Proof. Since $r_0 \notin N_G(P)$, let C be the connected component of $G \setminus N_G(P)$ that contains r_0. Let $N = N_G(C)$. Clearly, $N \subseteq N_G(P)$ and there exists an inclusion-minimal separator $S \subseteq N$ separating r_0 from x and y. Let u (resp., v) be a vertex of S in the path between x (resp., y) and r_0 in T (see Fig. 2(c)). If G is chordal, S induces a clique since S is a minimal separator [Gol04], therefore $\{u,v\} \in E(G)$. Finally, $\{u,v\} \notin E(T)$ because the opposite would imply that u or v is the smallest common ancestor of x and y, i.e., $r_0 \in \{u,v\}$, a contradiction since $u, v \in S$.

Lemma 3. *Let $k \geq 3$. Let G be a k-chordal graph and let T be a spanning tree defined by any BFS ordering. Then, for any $x, y \in V(G)$, $Str(\mathcal{R}(G,T),xy) \leq k-1$.*

Proof. By the previous subsection, it remains to prove the following case: r_0 is the smallest common ancestor of x and y, and some shortest path P between

x and y has no internal vertex in $T_{x\to r_0} \cup T_{r_0\to y}$. Recall that, if the route R_{xy} computed by $\mathcal{R}(G,T)$ takes a shortcut, this edge is denoted $\{e,f\}$. There are two cases to be considered.

We first assume that r_0 is not in the neighborhood of P, $N_G(P)$ (cf., Figure 2(d)).

Let us choose u and v as defined in Claim 3.2 and such that $d_G(r_0,u) + d_G(v,r_0)$ is minimum. Let $u' \in N_G(u) \cap P$ and $v' \in N_G(v) \cap P$ such that $d_G(u',v')$ is minimum. Let $P = P_1 \cup P_2 \cup P_3$, where P_1 is the subpath of P between x and u', P_2 is the subpath of P between u' and v', and P_3 is the subpath of P between v' and y. In the following we assume that u' is between v' and x in P. Otherwise, the proof is similar by setting P_1 is the subpath of P between x and v', P_2 is the subpath of P between v' and u', and P_3 is the subpath of P between u' and y.

Because T is a Strong BFS-tree, $|T_{x\to u}| \leq 1 + |P_1|$ and $|T_{v\to y}| \leq 1 + |P_3|$.

– First, let us assume that $e \in T_{r_0\to u}$ and $f \in T_{v\to r_0}$ (possibly $e = f = r_0$).
In particular, this means that there are no edges between a vertex in $T_{e\to u}$ and $T_{v\to f}$ but $\{e,f\}$. Therefore, by the choice of u,u',v,v', the cycle $C = \{u,u'\} \cup P_2 \cup \{v',v\} \cup T_{v\to f} \cup \{e,f\} \cup T_{e\to u}$ has no chord. Thus, $|C| = 3 + |P_2| + |T_{v\to f}| + |T_{e\to u}| \leq k$.
It follows that $|R_{xy}| = |T_{x\to u}| + |T_{u\to e}| + 1 + |T_{f\to v}| + |T_{v\to y}| \leq k - |P_2| + |P_1| + |P_3|$.
If $|P_2| > 0$, $|R_{xy}| - |P| = Str(\mathcal{R}(G,T), xy) \leq k - 1$.
Therefore, let us consider the case when $|P_2| = 0$, i.e., $u' = v'$. We first consider the case when $u > v$ (in the BFS ordering defining T). We aim at proving that $|T_{v\to y}| \leq |P_3|$. For purpose of contradiction, let us assume that $|T_{v\to y}| = |P_3| + 1$. Let w_1 be the child of v in $T_{v\to y}$. For any $1 \leq i \leq |P_3| + 1$ and let u_i be the i^{th} vertex on the path P_3 ($u' = u_1$), and let w_i be the i^{th} vertex on the path $T_{w_1\to y}$. Note that, because $|T_{w_1\to y}| = |P_3|$, $u_{|P_3|+1} = w_{|P_3|+1} = y$. Because u' is adjacent to u and $u > v$, it follows that $u' > w_1$. By a trivial induction on $i \leq |P_3|$, we get that, for any $1 \leq i \leq |P_3|$, $u_i > w_i$. Moreover, $w_{|P_3|}$ and $u_{|P_3|}$ are in the same layer, they are both adjacent to y and $u_{|P_3|} > w_{|P_3|}$. Hence, $\{w_{|P_3|}, y\}$ cannot belong to T, a contradiction.
Therefore, $|T_{v\to y}| \leq |P_3|$. Hence, $|R_{xy}| = |T_{x\to u}| + |T_{u\to e}| + 1 + |T_{f\to v}| + |T_{v\to y}| \leq k - |P_2| + |P_1| + |P_3| - 1 = k + |P_1| + |P_3| - 1 \leq |P| + k - 1$.
Now, let us consider the case when $u < v$. We prove that R_{yx} (the computed route from y to x) has length at most $|P| + k - 1$. By Lemma 1, it proves that $|R_{xy}| \leq |P| + k - 1$. If $R_{yx} = R_{xy}$, the proof is similar to the previous one (by symmetry). Therefore, let us assume that $R_{yx} \neq R_{xy}$. By Lemma 1, R_{yx} uses a shortcut $\{e',f'\} \in E(G) \setminus E(T)$ such that e' is the parent of e and f' is a child of f. If $e' \in T_{r_0\to u}$ and $f' \in T_{v\to r_0}$, again, the proof is similar to the previous one by symmetry. Hence, the only remaining case is $e' \in T_{r_0\to u} \setminus \{u\}$, f' is the child of $v = f$ (because of the relative positions of e,e',f,f',u,v). This case is similar (by symmetry) to the case treated in the third item of this proof.

- Second, let us assume that $e \in T_{x\to u} \setminus \{u\}$ and $f \in T_{v\to y}$ In this case, $|R_{xy}| = |T_{x\to e}| + 1 + |T_{f\to y}| \leq |T_{x\to u}| + |T_{v\to y}| \leq 2 + |P_1| + |P_3| \leq |P| + k - 1$ (because $k \geq 3$).
  The case $e \in T_{x\to u}$ and $f \in T_{v\to y} \setminus \{v\}$ is symmetric. Indeed, in this case, consider $\{e', f'\}$ the shortcut used by R_{yx}. By Lemma 1, $f' \in T_{y\to v} \setminus \{v\}$ and $e \in T_{x\to u}$. Hence, applying the same proof to R_{yx}, $|R_{xy}| = |R_{yx}| \leq |P| + k - 1$.
- Finally, let us assume that $e \in T_{x\to u} \setminus \{u\}$ and $f \in T_{v\to r_0} \setminus \{v\}$ (cf., Figure 2(d)). In this case, let us study the layers of e, f and y.
  First, $\ell(e) = \ell(u) + |T_{u\to x}| - |T_{e\to x}|$. Because $\{e, f\} \in E(G)$ and T is a Strong BFS-tree, $\ell(e) - 1 \leq \ell(f) \leq \ell(e) + 1$. Besides, $\ell(y) = \ell(f) + |T_{f\to v}| + |T_{v\to y}|$, and, because $P_2 \cup P_3$ is a shortest path, $\ell(y) \leq \ell(u') + |P_2| + |P_3| \leq \ell(u) + 1 + |P_2| + |P_3|$.
  Therefore, $\ell(u) + 1 + |P_2| + |P_3| \geq \ell(u) + |T_{u\to x}| - |T_{e\to x}| - 1 + |T_{f\to v}| + |T_{v\to y}|$.
  Hence, $2 + |P_2| + |P_3| \geq |T_{u\to x}| - |T_{e\to x}| + |T_{f\to v}| + |T_{v\to y}|$.
  Now, $|R_{xy}| = |T_{e\to x}| + 1 + |T_{f\to v}| + |T_{v\to y}| \leq |P_2| + |P_3| - |T_{u\to x}| + 2|T_{e\to x}| + 3 = |P_2| + |P_3| + |T_{u\to x}| - 2(|T_{u\to x}| - |T_{e\to x}|) + 3 \leq |P_1| + |P_2| + |P_3| + 2 \leq |P| + k - 1$.
  Again, the case $e \in T_{u\to r_0} \setminus \{u\}$ and $f \in T_{v\to y} \setminus \{v\}$ is symmetric.

To conclude, let us assume that $r_0 \in N_G(P)$ (cf., Figure 2(e)). Let $z \in N_P(r_0)$. Let $P = P_1 \cup P_2$ where P_1 is the subpath of P between x and z, and P_2 is the subpath of P between z and y. Because T is a Strong BFS-tree, $|T_{x\to r_0}| \leq 1 + |P_1|$ and $|T_{r_0\to y}| \leq 1 + |P_2|$. Therefore, $|R_{xy}| \leq |T_{x\to r_0}| + |T_{r_0\to y}| \leq |P_1| + |P_2| + 2 \leq |P| + k - 1$ (because $k \geq 3$).

It is important to note that the previous result is valid whatever Strong BFS-tree is used. However, it is easy to observe that the inequality given by Lemma 3 is optimal. Indeed, Figure 1(b) represents a k-chordal graph with $k = 2p + 2$ ($p \geq 1$) and a Strong BFS-tree T (that actually is a MaxBFS-tree) such that $Str(\mathcal{R}(G, T))) = 2p + 1 = k - 1$: a message from 1 to 2 will pass through the edge $\{3p + 3, 3p + 4\}$.

Lemma 3 gives that, for any chordal graph G and for any Strong BFS-tree T, $Str(\mathcal{R}(G, T)) \leq 2$. The following lemma proves that we can improve a bit the stretch in case of a chordal graph by using a "better" BFS-tree, i.e., a MaxBFS-tree.

Lemma 4. *Let G be a chordal graph and let T be a spanning tree defined by any MaxBFS ordering. Then, for any $x, y \in V(G)$, $Str(\mathcal{R}(G, T), xy) \leq 1$.*

Sketch of the Proof. Due to lack of space, the proof is sketched and the full proof can be found in [NSR]. Again, it only remains to consider the case when r_0 is the smallest common ancestor of x and y, and some x-y-shortest path P in G has no internal vertex in $T_{x\to r_0} \cup T_{r_0\to y}$.

If $r_0 \notin N_G(P)$ (cf., Figure 2(c)), let u and v be defined as in Claim 3.2. $\{u, v\} \in E(G)$ because G is chordal. Hence, we prove that u and v have a common neighbor z in P. If $z \in \{x, y\}$, we prove that either $|T_{v\to y}| \leq |P|$ or x is adjacent to the child of v in $T_{v\to y}$, and in both cases $Str(\mathcal{R}(G, T), xy) \leq 1$. Otherwise, let P_1 be the subpath of P between x and z, and let P_2 be the

subpath of P between z and y. By symmetry, w.l.o.g., assume $u > v$ (otherwise, the same proof holds for R_{yx}). Because $u > v$, we have $|T_{v\to y}| \leq |P_2|$. Finally, we prove that either $|T_{x\to u}| \leq |P_1|$, or $|T_{x\to u}| = |P_1|+1$ and v is adjacent to the child w of u in $T_{u\to x}$. Indeed, if $|T_{x\to u}| = |P_1|+1$, we prove that $\{w, z\} \in E(G)$ (by chordality) and $u > v > w > z$. $\{v, w\} \in E(G)$ follows because we consider a MaxBFS ordering and by Theorem 2. It is then easy to conclude because $|R_{xy}| \leq |T_{x\to u}| + 1 + |T_{v\to y}|$.

If $r_0 \in N_G(P)$, the proof shows that either $|T_{x\to r_0}| < |P_1| + 1$ or $|T_{r_0\to y}| < |P_2| + 1$. □

It is easy to observe that the inequality given by Lemma 4 is optimal. Indeed, consider Figure 1(c) and the route between 1 and 2. The above discussion and lemmata prove Theorem 3.

4 Distributed Algorithm

In this section, we present a simple distributed algorithm that computes the routing tables sufficient for the execution of the routing scheme described in the previous section. For space restrictions, let us give just an informal description. The algorithm consist of three phases. The first two of them aim at building a Strong BFS-tree T. Then, during the third phase each vertex x is assigned an integral label $P(x)$ that corresponds to its position in a DFS postorder traversal of T. It is easy to check that it gives a Strong BFS-ordering of G. Moreover, x learns $I(x)$, the interval corresponding to the labels of its descendants in T (including x). $P(x)$ is used as the identifier of x in the routing scheme. At each vertex y, for every neighbor x of y (except for the parent of y), the edge yx is labeled with $I(x)$. Let us describe the algorithm in detail.

1^{st} **Phase.** The first phase chooses an arbitrary vertex $r \in V(G)$ as the root and gives to each vertex its *layer*, i.e. its distance from r. Moreover, each vertex informs its neighbors of its own layer. This trivially takes at most $D+1$ steps by broadcasting a counter initially set to 1 by the root. Now, if each vertex chooses an arbitrary neighbor in the lower layer as the parent, the obtained graph is a BFS-tree. However, as soon as Strong BFS-trees or MaxBFS-tree are concerned, not any neighbor in the lower layer can be chosen as the parent.

2^{nd} **Phase.** The second phase aims at determining an appropriate parent for each vertex. For this purpose, we assign an ordering on the vertices based on the following labeling: the root receives an empty label and any vertex $v \in V(G)$ in the layer $i \geq 1$ will eventually have a full label $label(v)$, where $label(v)$ is a sequence of i integers that consists of the full label of its parent u concatenated with the integer p that indicates that v is the p^{th} child of u. The labels will be constructed gradually, in a way that each vertex will be aware of the current (partial) labels of its neighbors. Notice that the lexicographic ordering of full labels gives the inverse of the Strong BFS-ordering (or MaxBFS-ordering) under construction. Transforming it into integer numbers ranging from n down to 1 can be easily computed once we have fixed T and ordered the children of each node (see the third phase).

To see how the algorithm assigns full labels, let us assume that, at some step, each vertex in layers up to $i-1$ has received his full label as defined previously. Moreover, assume that each vertex in layer i knows the full labels of its neighbors in layer $i-1$ and the partial labels of its neighbors in layer i. In particular, each vertex v in layer i knows its neighbor in layer $i-1$ with the smallest label in the lexicographical ordering. So v can choose this node as its parent and inform all its neighbors of its choice. Once each vertex in layer i has chosen a parent, the vertices in layer $i-1$ establish an ordering on their children: any vertex u in layer $i-1$ sends an integer p_v to its child v so that v knows it is the p_v^{th} child of u (see below). This implies that the induction condition holds for layers i and $i+1$. Notice that at layers 0 and 1 the condition trivially holds, since layer 0 contains a single vertex, the root r, with an empty label.

Spreading of labels. Let us describe how to spread the labels of the vertices efficiently. For any $i \geq 1$, each vertex v in the layer i maintains a subset $PP(v)$ (for potential parent) of its neighbors in layer $i-1$ that initially contains all these neighbors. Once a vertex v in layer i has received a label with integer p_v (that corresponds to its position among its siblings), it transmits the pair (i, p_v) to all its neighbors. Once a vertex v in the layer $j > i$ has received such a message (i, p) from all of its neighbors u in $PP(v)$, it keeps in $PP(v)$ the vertices that have the smallest p. Then, v transmits the corresponding pair to all its neighbors. Moreover, receiving such a message from any neighbor u', v adds p to the locally stored (partial) label of u'. Proceeding in this way, once any vertex v in layer i has received a label, any vertex in layer $i+1$ knows its potential parents, i.e., its neighbors in $PP(v)$, and the corresponding label.

Ordering of children in Strong BFS-tree. Once each vertex in layer i has chosen a parent, the vertices in layer $i-1$ establish an ordering on their children. If we want to obtain a Strong BFS-tree without additional properties, any ordering is valid. Therefore, each vertex in the layer $i-1$ arbitrarily orders its children and sends them their position in this ordering. Then, each vertex in layer i has a full label. This takes one step per layer, i.e., this takes time $O(D)$ in total.

Ordering of children in MaxBFS-tree. In this case, each vertex in layer $i-1$ will order its children according to the number of neighbors with smaller labels they have. In other words, each vertex in layer $i-1$ orders its children according to the number of their neighbors that will have larger numbers in the final ordering. Notice that as soon as the vertices of layer i have chosen their parent and broadcasted them to their neighbors, a vertex v in layer i only needs to learn its position in the ordering relatively to its siblings in T. Therefore, children of different vertices from layer $i-1$ can be ordered in parallel.

A vertex u in layer $i-1$ orders its children as follows. Let us assume u has already ordered its first p neighbors ($p \geq 0$). These neighbors of u have full label while remaining neighbors of u only have partial label. Vertex u chooses its $p+1^{th}$ child v as the one with the greatest number of neighbors that either have a parent greater than u or that are siblings of v with a full label. v receives $p+1$

from u and completes its label (that becomes full). Then, v informs its siblings that it has received a full label, and each child of u updates the number of its neighbors that already have full label. In this way, u orders its $d(u)$ children in $O(d(u))$ time. So, in total, this step is executed in $O(\Delta)$ time per layer. Therefore, in at most $O(\min\{\Delta D, n\})$ steps, any vertex has chosen a unique vertex as its parent and the tree T rooted in r is well defined.

3^{rd} **Phase.** The third phase consists in assigning to each vertex v his position in the ordering and the interval of positions of vertices that belong to T_v, the subtree of T rooted in v. It is easy to do so by two stages, the first one consisting in propagation of messages from the leaves toward the root and the second one from the root toward the leaves. During the first stage, the leaves of T send 1 to their parents, and any vertex u with children $v_1, \cdots, v_r$ receives from v_i $(1 \leq i \leq r)$ the number ℓ_i of vertices belonging to the subtree of T rooted in v_i and sends to its parent $1 + \sum_{i \leq r} \ell_i$. During the second stage, the root is assigned the position n and the interval $[1, \ldots, |V(G)|]$; each vertex v takes the last position in the interval it has received and partitions the rest into subintervals corresponding to each of its children. It is easy to check that the resulting ordering corresponds to a DFS postorder traversal of T. This phase takes at most $2D$ steps. The discussion of this section can be summarized with the following theorem.

Theorem 4. *In any n-node network G with diameter D and maximum degree Δ, the distributed protocol described above computes routing tables of $O(\Delta \log n)$ bits per node, for the execution of the routing scheme $\mathcal{R}(G, T)$. Our protocol is executed in time $O(D)$ with $O(logn)$-bit messages if the desired tree T is an arbitrary Strong BFS-tree, and in time $O(\min\{\Delta D, n\})$ if T is a MaxBFS-tree.*

5 Open Problems

Many questions remain open in this study. In particular, is it possible to design a routing scheme achieving same stretch and time-complexity but using smaller routing tables? Which stretch can we achieve when few large cycles are allowed? Routing schemes in dynamic networks, i.e., when nodes are free to leave or to arrive in the network at any time, are needed. Fault-tolerant and self stabilizing algorithms to compute routing tables would be appreciated.

References

[AG06] Abraham, I., Gavoille, C.: Object location using path separators. In: 25th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 188–197 (2006)

[AGGM06] Abraham, I., Gavoille, C., Goldberg, A.V., Malkhi, D.: Routing in networks with low doubling dimension. In: 26th IEEE International Conference on Distributed Computing Systems (ICDCS), p. 75 (2006)

[BKS05] Berry, A., Krueger, R., Simonet, G.: Ultimate generalizations of lexbfs and lex m. In: 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 199–213 (2005)

[CK] Corneil, D.G., Krueger, R.: A unified view of graph searching. SIAM Journal on Computing (SICOMP)

[DG02] Dourisboure, Y., Gavoille, C.: Improved compact routing scheme for chordal graphs. In: 16th International Conference on Distributed Computing (DISC), pp. 252–264 (2002)

[Dou05] Dourisboure, Y.: Compact routing schemes for generalised chordal graphs. Journal of Graph Algorithms and Applications (JGAA) 9(2), 277–297 (2005)

[Dra05] Dragan, F.F.: Estimating all pairs shortest paths in restricted graph families: a unified approach. Journal of Algorithms 57(1), 1–21 (2005)

[FG65] Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pacific Journal of Mathematics 15, 835–855 (1965)

[FG01] Fraigniaud, P., Gavoille, C.: Routing in trees. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 757–772. Springer, Heidelberg (2001)

[Fra05] Fraigniaud, P.: Greedy routing in tree-decomposed graphs. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 791–802. Springer, Heidelberg (2005)

[Gav00] Gavoille, C.: A survey on interval routing. Theoretical Computer Science (TCS) 245(2), 217–253 (2000)

[GG01] Gavoille, C., Gengler, M.: Space-efficiency for routing schemes of stretch factor three. Journal of Parallel and Distributed Computing (JPDC) 61(5), 679–687 (2001)

[Gol04] Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs (2004)

[GP96] Gavoille, C., Perennes, S.: Memory requirements for routing in distributed networks (extended abstract). In: 15th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 125–133 (1996)

[GP99] Gavoille, C., Peleg, D.: The compactness of interval routing. SIAM Journal on Discrete Mathematics (SIDMA) 12(4), 459–473 (1999)

[GP01] Gavoille, C., Peleg, D.: The compactness of interval routing for almost all graphs. SIAM Journal on Computing (SICOMP) 31(3), 706–721 (2001)

[NN98] Narayanan, L., Nishimura, N.: Interval routing on k-trees. Journal of Algorithms 26(2), 325–369 (1998)

[NSR] Nisse, N., Suchan, K., Rapaport, I.: Distributed computing of efficient routing schemes in generalized chordal graphs, http://www-sop.inria.fr/members/Nicolas.Nisse/publications/distribRouting.pdf

[PU89] Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. Journal of the ACM 36(3), 510–530 (1989)

[SK85] Santoro, N., Khatib, R.: Labelling and implicit routing in networks. The Computer Journal 28(1), 5–8 (1985)

[Tho04] Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. Journal of the ACM 51(6), 993–1024 (2004)

[TZ01] Thorup, M., Zwick, U.: Compact routing schemes. In: 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 1–10 (2001)

A Versatile STM Protocol with Invisible Read Operations That Satisfies the Virtual World Consistency Condition

Damien Imbs and Michel Raynal

IRISA, Université de Rennes 1, 35042 Rennes, France
{damien.imbs,raynal}@irisa.fr

Abstract. The aim of a Software Transactional Memory (STM) is to discharge the programmers from the management of synchronization in multiprocess programs that access concurrent objects. To that end, a STM system provides the programmer with the concept of a transaction. The job of the programmer is to design each process the application is made up of as a sequence of transactions. A transaction is a piece of code that accesses concurrent objects, but contains no explicit synchronization statement. It is the job of the underlying STM system to provide the illusion that each transaction appears as being executed atomically. Of course, for efficiency, a STM system has to allow transactions to execute concurrently. Consequently, due to the underlying STM concurrency management, a transaction commits or aborts.

This paper first presents a new STM consistency condition, called *virtual world* consistency. This condition states that no transaction reads object values from an inconsistent global state. It is similar to opacity for the committed transactions but weaker for the aborted transactions. More precisely, it states that (1) the committed transactions can be totally ordered, and (2) the values read by each aborted transaction are consistent with respect to its causal past only. Hence, virtual world consistency is weaker than opacity while keeping its spirit. Then, assuming the objects shared by the processes are atomic read/write objects, the paper presents a STM protocol that ensures virtual world consistency (while guaranteeing the invisibility of the read operations). From an operational point of view, this protocol is based on a vector-clock mechanism. Finally, the paper considers the case where the shared objects are regular read/write objects. It also shows how the protocol can easily be weakened while still providing an STM system that satisfies *causal consistency*, a condition strictly weaker than virtual world consistency.

Keywords: Atomic object, Causal past, Commit/abort, Concurrency control, Consistency condition, Consistent global state, Lock, Read-from relation, Regular Read/write object, Serializability, Shared memory, Software transactional memory, Vector clock, Transaction.

1 Introduction

The challenging advent of multicore architectures. The speed of light has a limit. When combined with other physical and architectural demands, this physical constraint places

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 266–280, 2010.

limits on processor clocks: their speed cannot be further be incremented. Hence, software performance can no longer be obtained by increasing CPU clock frequencies. To face this new challenge, (since a few years ago) manufacturers have investigated and are producing what they call *multicore architectures*, i.e., architectures in which each chip is made up of several processors that share a common memory. This constitutes what is called "the multicore revolution" [12].

The main challenge associated with multicore architectures is "how to exploit their power?" Of course, the old (classical) "multi-process programming" (multi-threading) methods are an answer to this question. Basically, these methods provide the programmers with the concept of a *lock*. According to the abstraction level considered, this lock can be a semaphore object, a monitor object, or more generally the base synchronization object provided by the underlying programming language.

Unfortunately, traditional lock-based solutions have inherent drawbacks. On one side, if the set of data whose accesses are controlled by a single lock is too large (large grain), the parallelism can be drastically reduced. On another side, the solutions where a lock is associated with each datum (fine grain), are error-prone (possible presence of subtle deadlocks), difficult to design, master and prove correct. In other words, providing the application programmers with locks is far from being the panacea when one has to produce correct and efficient multi-process (multi-thread) programs. Interestingly enough, multicore architectures have (in some sense) rang the revival of concurrent programming.

The Software Transactional Memory approach. The concept of *Software Transactional Memory* (STM) is an answer to the previous challenge. The notion of transactional memory has first been proposed (fifteen years ago) by Herlihy and Moss to implement concurrent data structures [13]. It has then been implemented in software by Shavit and Touitou [24], and has recently gained a great momentum as a promising alternative to locks in concurrent programming, e.g., [9,11].

Transactional memory abstracts the complexity associated with concurrent accesses to shared data by replacing locking with atomic execution units (called transactions). In that way, the programmer has to focus where atomicity is required and not on the way it has to be realized. The aim of a STM system is consequently to discharge the programmer from the direct management of synchronization entailed by accesses to concurrent objects.

More generally, STM is a middleware approach that provides the programmers with the *transaction* concept (this concept is close but different from the notion of transactions encountered in databases [9]). More precisely, a process is designed as (or decomposed into) a sequence of transactions, each transaction being a piece of code that, while accessing any number of shared objects, always appears as being executed atomically. The job of the programmer is only to define the units of computation that are the transactions. He does not have to worry about the fact that the objects can be concurrently accessed by transactions. Except when he defines the beginning and the end of a transaction, the programmer is not concerned by synchronization. It is the job of the STM system to ensure that transactions execute as if they were atomic.

Of course, a solution in which a single transaction executes at a time trivially implements transaction atomicity but is irrelevant from an efficiency point of view. So, a STM

system has to do "its best" to execute as many transactions per time unit as possible. Similarly to a scheduler, a STM system is an on-line algorithm that does not know the future. If the STM is not trivial (i.e., it allows several transactions that access the same objects in a conflicting manner to run concurrently), this intrinsic limitation can direct it to abort some transactions in order to ensure both transaction atomicity and object consistency. From a programming point of view, an aborted transaction has no effect (it is up to the process that issued an aborted transaction to re-issue it or not; usually, a transaction that is restarted is considered as a new transaction).

Content of the paper and roadmap. This paper is made up of 5 sections and has three contributions. Section 2 presents the computation model and the first contribution, namely, a new consistency condition, called *virtual world* consistency. Differently from serializability but similarly to opacity, this condition (1) takes into account both the committed transactions and the aborted transactions, but (2) is strictly weaker than opacity (and can consequently allow more transactions to commit). Intuitively, both opacity and virtual world consistency requires that every transaction (whatever its fate, commit or abort) reads object values from a consistent global state. They differ in what each considers as a *consistent* global state.

The second contribution, namely, a STM protocol that satisfies virtual world consistency, is presented in Section 3. Among its noteworthy features, this protocol allows invisible read operations (i.e., when a transaction reads an object, it is not required to write control information into the shared memory to inform the other transactions on possible read/write conflicts). From an operational point of view, the protocol does not use a global logical clock, but a distributed vector clock with one entry per object. So, the protocol is targeted for applications that manipulate few shared objects.

Then, Section 4 addresses the versatility of the proposed STM protocol (third contribution). It shows that the simple suppression of a consistency check provides a protocol that ensures the *causal consistency* condition. It also shows that, with minimal modification, the protocol may ensure virtual world consistency under the use of *regular* instead of the stronger *atomic* objects shared. Finally, Section 5 concludes the paper.

2 A STM Computation Model

2.1 Why a Consistency Condition Has to Take into Account the Aborted Transactions

The classical consistency criterion for database transactions is serializability [21] (sometimes strengthened in "strict serializability", as implemented when using the 2-phase locking mechanism). The serializability consistency criterion involves only the transactions that commit. Said differently, a transaction that aborts is not prevented from accessing an inconsistent state before aborting. Differently from database transactions that are usually produced by SQL queries, in a STM system the code encapsulated in a transaction is not restricted to particular patterns. Consequently a transaction always has to operate on a consistent state. To be more explicit, let us consider the following example where a transaction contains the statement $x \leftarrow a/(b-c)$ (where a, b and c are integer data), and let us assume that $b-c$ is different from 0 in all the consistent

states (intuitively, a consistent state is a global state that, considering only the committed transactions, could have existed at some real time instant). If the values of b and c read by a transaction come from different states, it is possible that the transaction obtains values such as $b = c$ (and $b = c$ defines an inconsistent state). If this occurs, the transaction throws an exception that has to be handled by the process that invoked the corresponding transaction. (Even worse undesirable behaviors can be obtained when reading values from inconsistent states. This occurs for example when an inconsistent state provides a transaction with values that generate infinite loops.) Such bad behaviors have to be prevented in STM systems: whatever its fate (commit or abort) a transaction has to always see a consistent state of the data it accesses. The aborted transactions have to be harmless. This observation has first been stated in [8].

2.2 From Opacity to Virtual World Consistency

Opacity. Informally suggested in [8], and formally introduced and investigated in [10], the *opacity* consistency condition requires that no transaction reads values from an inconsistent global state where, considering only the committed transactions, a *consistent global state* is defined as the state of the shared memory at some real time instant. Opacity is the same as strict serializability when we consider all the committed transactions, plus an appropriate read prefix for each aborted transaction.

More precisely, let us associate with each aborted transaction T its execution prefix (called *read prefix*) that contains all its read operations until T aborts (if the abort is entailed by a read, this read is not included in the prefix). An execution of a set of transactions satisfies the *opacity* condition if all the committed transactions plus the read prefix of each aborted transaction appear as if they have been executed one after the other (this is a "witness sequential execution"), this witness sequential execution being in agreement with the real time occurrence order of each transaction. (Examples of protocols implementing the opacity property – each with different additional features – can be found in [8, 15, 16, 23].)

Virtual world consistency. This consistency condition is weaker than opacity while keeping its spirit. It states that (1) no transaction (committed or aborted) reads values from an inconsistent global state, (2) the consistent global states read by the committed transactions are mutually consistent (in the sense that they can be totally ordered) but (3) while the global state read by each aborted transaction is consistent from its individual point of view, the global states read by any two aborted transactions are not required to be mutually consistent. Said differently, virtual world consistency requires that (1) all the committed transactions be serializable [21] (so they all have the same "witness sequential execution") or linearizable [14] (if we want this witness execution to also respect real time) and (2) each aborted transaction (reduced to a read prefix as explained previously) reads values that are consistent with respect to its causal past only[1]. As two aborted transactions can have different causal pasts, each can read from a global state

[1] The notion of *causal past* of a transaction is analogous to the notion of causal past encountered in message-passing [5, 26]. See [18] for a formal definition and a parallel between transaction systems and message-passing systems.

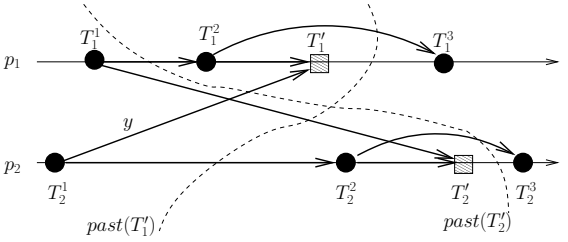


Fig. 1. Examples of causal pasts

that is consistent from its causal past point of view, but these two global states can be mutually inconsistent as aborted transactions have not necessarily the same causal past (hence the name *virtual world* consistency). This consistency condition can benefit lots of STM applications as, from its local point of view, a transaction cannot differentiate it from opacity.

The formal definition of virtual world consistency is based on a partial order on the committed transactions and a partial order on the whole set of transactions (where each aborted transactions is reduced to an appropriate read prefix). To be precise and unambiguous, this definition requires a few pages. Due to page limitation, the condition is explained here informally with a simple example. The reader is referred to [18] for the formal definition. Let us consider the transaction execution depicted on Figure 1. There are two processes: p_1 has sequentially issued T_1^1, T_1^2, T_1' and T_1^3, while p_2 has issued T_2^1, T_2^2, T_2' and T_2^3. The transactions associated with a black dot have committed, while the ones with a grey square have aborted. From a dependency point of view, each transaction issued by a process depends on its previous committed transactions (process order relation[2]), and on committed transactions issued by the other process as defined by the read-from relation due to the accesses to the shared objects, (e.g., the label y on the dependency edge from T_2^1 to T_1' means that T_1' has read fro y a value written by T_2^1). Differently, since an aborted transaction does not write shared objects, there is no dependency edges originating from it. The causal past of the aborted transactions T_1' and T_2' are indicated on the figure (left of the corresponding dotted lines). Virtual world consistency requires the following: (1) the committed transactions are serializable (or strict serializable if we want the witness sequence to respect the additional real time order constraint), and (2) each aborted transaction reads values from a state that is consistent with respect to its causal past (as an example, the values read by T_1' are consistent wrt the dependencies as indicated on Figure 1).

That consistency condition actually extends to STM systems the notions of *consistent cut*, *causal past*, and *consistent global state* encountered in asynchronous message-passing systems [5, 6, 7, 26]. In these systems, two different processes can simultaneously compute two global states such that each global state is consistent with respect to the causal past of the invoking process, but these global states are mutually inconsistent from the point of view of an external omniscient sequential observer (i.e., they cannot

[2] A process issues a new transaction only when its previous transaction has completed (by committing or aborting). This defines the *process order* relation [18].

be serialized). The "read-from" relation linking transactions is the STM equivalent of the "message" relation that defines the flow of information exchange in message-passing systems. The "process order relation" is the same as in message-passing systems.

In addition to the fact that it can allow more transactions to commit than opacity, one of the main interests of virtual world consistency lies in the fact that it prevents bad phenomena (as described in Section 2.1) from occurring without requiring all the transactions (committed or aborted) to agree on the same witness execution. Let us assume that, when executed alone and it reads a consistent state of the objects, each transaction behaves correctly (e.g. it does not entail a division by 0, does not enter an infinite loop, etc.). As, due to the virtual world consistency condition, no transaction (committed or aborted) reads from an inconsistent state, it cannot behave incorrectly despite concurrency; it can only be aborted. This is a first class requirement for transactional memories.

2.3 The STM System Interface

The STM system provides the transactions with four operations denoted $\mathsf{begin}_T()$, $X.\mathsf{read}_T()$, $X.\mathsf{write}_T()$, and $\mathsf{try\_to\_commit}_T()$, where T is a transaction, and X a shared base object.

- $\mathsf{begin}_T()$ is invoked by T when it starts. It initializes local control variables.
- $X.\mathsf{read}_T()$ is invoked by the transaction T to read the base object X. That operation returns a value of X or the control value $abort$. If $abort$ is returned, the invoking transaction is aborted (in that case, the corresponding read does not belong to the read prefix associated with T).
- $X.\mathsf{write}_T(v)$ is invoked by the transaction T to update X to the new value v. That operation returns the control value ok or the control value $abort$. In the proposed protocol it always returns ok.
- If a transaction attains its last statement (as defined by the user, which means it has not been aborted before) it executes the operation $\mathsf{try\_to\_commit}_T()$. That operation decides the fate of T by returning $commit$ or $abort$. (Let us notice, a transaction T that invokes $\mathsf{try\_to\_commit}_T()$ has not been aborted during an invocation of $X.\mathsf{read}_T()$.)

2.4 The Incremental Read + Deferred Update Model

In this transaction system model, each transaction T uses a local working space. When T invokes $X.\mathsf{read}_T()$ for the first time, it reads the value of X from the shared memory and copies it into its local working space. Later $X.\mathsf{read}_T()$ invocations (if any) use this copy. So, if T reads X and then Y, these reads are done incrementally, and the state of the shared memory may have changed in between. One usually says that the transaction T computes an *incremental snapshot*[3].

[3] The incremental approach to compute a snapshot reads asynchronously (separately) one object after the other. Differently, in [2,4,17], the whole set of the base objects to be atomically read is globally defined at the time of the snapshot invocation.

When T invokes $X.\mathsf{write}_T(v)$, it writes v into its working space (and does not access the shared memory). Finally, if T is not aborted while it is executing $\mathsf{try\_to\_commit}_T()$, it copies the values written (if any) from its local working space to the shared memory. (A similar deferred update model is used in some database transaction systems.)

2.5 Processes and Atomic Base Objects

The system is made up of an arbitrary number of processes and m base shared objects. The processes are denoted p_i, p_j, etc., while the objects are denoted $X, Y, \ldots$, where each id X is such that $X \in \{1, \cdots, m\}$. Each process consists of a sequence of transactions (that are not known in advance).

Each of the m base objects is an atomic read/write object [20]. This means that the read and write operations issued on such an object X appear as if they have been executed sequentially, and this "witness sequence" is legal (a read returns the value written by the closest write that precedes it in this sequence) and respects the real time occurrence order on the operations on X (if $op1(X)$ is terminated before $op2(X)$ starts, $op1$ appears before $op2$ in the witness sequence).

3 A STM Protocol When the Base Objects Are Atomic

3.1 The STM Algorithm: Control Variables

On a base object side. Each base atomic object X is made up of two fields: $X.value$ which contains its value, and a vector $X.depend[1..m]$ that tracks value dependencies. More precisely, $X.depend[X]$ is the sequence number of the current value of X, while $X.depend[Y]$ ($Y \neq X$) is the sequence number of the value of Y on which the current value of X depends. (A sequence number can be seen as a logical date associated with an object.) Moreover a lock is associated with every base object.

On a process side. A process issues transactions sequentially. So, when a process p_i issues a new transaction, that transaction has to work with object values that are not older than the ones used by the previous transactions issued by p_i. To that end, p_i manages a local vector $p\_depend_i[1..m]$ such that $p\_depend_i[X]$ contains the sequence number of the last value of X that (directly or indirectly) is known by p_i.

In addition to the previous array whose scope is the lifetime of the corresponding process, a process p_i manages local variables whose scope is the one of its current transaction T. Those are:
- An array $t\_depend_T[1..m]$ that is used instead of $p\_depend_i[1..m]$ during the execution of T. This is necessary because $p\_depend_i[1..m]$ must not be modified if T aborts,
- A set lrs_T (resp., lws_T) that is the read set (resp., write set) of the transaction T currently executed by p_i,
- Finally, for every object X accessed by T, p_i keeps a local copy that is denoted $lc(X)$.

3.2 The STM Algorithm

The code of the STM system for a process p_i is described in Figure 2. It consists the algorithms that implement the four operations of the STM interface (Section 2.3), namely, $\mathsf{begin}_T()$, $X.\mathsf{read}_T()$, $X.\mathsf{write}_T()$, and $\mathsf{try\_to\_commit}_T()$, where T is a transaction issued by a process p_i and X a base object. When it is returned, the control value $abort$ is tagged 1 or 2 to indicate the cause of the abort to the corresponding transaction.

The operation $\mathsf{begin}_T()$. This operation is a simple initialization of the local control variables associated with the current transaction T. Let us notice that t_depend_T is initialized to p_depend_i to take into account the causal dependencies on the values previously accessed by p_i. This is due to the fact that a process p_i issues its transactions one after the other and the next one inherits the causal dependencies created by the previous ones.

The operation $X.\mathsf{read}_T()$. This operation returns a value of X or the control value $abort$ (in which case T is aborted). If (due to a previous read of X) there is a local copy, its value is returned (lines 01 and 07).

operation $\mathsf{begin}_T()$: $lrs_T \leftarrow \emptyset$; $lws_T \leftarrow \emptyset$; $t\_depend_T \leftarrow p\_depend_i$.
==========
operation $X.\mathsf{read}_T()$:
(01) **if (there is no local copy of** X**) then**
(02) allocate local space -denoted $lc(X)$- for a local copy of X; $lc(X) \leftarrow X$;
(03) $lrs_T \leftarrow lrs_T \cup \{X\}$; $t\_depend_T[X] \leftarrow lc(X).depend[X]$;
(04) **if** $(\exists Y \in lrs_T : t\_depend_T[Y] < lc(X).depend[Y])$ **then** $\mathsf{return}(abort, 1)$ **end if**;
(05) **for each** $Y \notin lrs_T$ **do** $t\_depend_T[Y] \leftarrow max(t\_depend_T[Y], lc(X).depend[Y])$ **end for**
(06) **end if**;
(07) return $(lc(X).value)$.
==========
operation $X.\mathsf{write}_T(v)$:
(08) **if** (there is no local copy of X) **then** allocate local space $lc(X)$ to store v **end if**;
(09) $lc(X).value \leftarrow v$; $lws_T \leftarrow lws_T \cup \{X\}$;return (ok).
==========
operation $\mathsf{try\_to\_commit}_T()$:
(10) **let** $ConsistencyCheck_T$ **be the predicate** $(\forall Z \in lrs_T : t\_depend_T[Z] = Z.depend[Z])$;
(11) lock all the objects in $lrs_T \cup lws_T$;
(12) **if** $(lrs_T \neq \emptyset)$ **then**
if $(\neg ConsistencyCheck_T)$ **then** release all the locks; $\mathsf{return}(abort, 2)$ **end if end if**;
(13) **if** $(lws_T \neq \emptyset)$ **then for each** $X \in lws_T$ **do** $t\_depend_T[X] \leftarrow X.depend[X] + 1$ **end for**;
(14) **for each** $X \in lws_T$ **do** $X \leftarrow (lc(X).value, t\_depend_T)$ **end for**
(15) **end if**;
(16) release all the locks;
(17) $p\_depend_i \leftarrow t\_depend_T$;
(18) $\mathsf{return}(commit)$.

Fig. 2. A STM algorithm that satisfies virtual world consistency

If $X.\mathsf{read}_T()$ is its first read of X, p_i first builds a copy $lc(X)$ from the shared memory (line 02), and updates accordingly its local control variables lrs_T and $t\_depend_T[X]$ (line 03).

As the reads are incremental (p_i does not read in one atomic action all the base objects it wants to read), p_i has to check that the value $lc(X).value$ it has just obtained from the shared memory and the values it has previously read can belong to a consistent global state. If it is not the case, p_i has to abort T, line 04. Let Y be an object that has been previously read by T. Let us observe that the sequence number of the value of Y read by T is kept in $t\_depend_T[Y]$. If the value of X just read by T depends on a more recent value of Y, the values of X and Y are mutually inconsistent. This is exactly what is captured by the predicate $\exists Y \in lrs_T : t\_depend_T[Y] < lc(X).depend[Y])$ (line 04). If this predicate is true, p_i aborts T. Otherwise, p_i first updates $t\_depend_T[1..m]$ (line 05) to take into account the new dependencies (if any) created by this reading of X, and finally returns the value obtained from X (line 07).

A $X.\mathsf{read}_T()$ operation is *visible* if the issuing transaction T has to write on shared memory to inform the other transactions on its read of X. Otherwise it is *invisible*.

Property 1. All the $X.\mathsf{read}_T()$ operations are invisible.

Property 2. If $(abort, 1)$ is returned to a transaction T, this is because T executes an operation $X.\mathsf{read}_T()$, and the abort is due to the fact that, while the values previously read by T belong to a consistent global state (also called "consistent snapshot"), the addition of the value of X obtained from the shared memory would make this snapshot inconsistent.

In the case of Property 2, the read prefix associated with the aborted transaction T contains the values read before the operation $X.\mathsf{read}_T()$, and does not contain the value read from X.

The operation $X.\mathsf{write}_T(v)$. The algorithm implementing that operation is very simple. If there is no local copy for the object X, one is created (line08). Then, the value v is written into that copy and the control variable lws_T is updated (line 09).

Property 3. No $X.\mathsf{write}_T()$ operation can entail the abort of a transaction.

The operation $\mathsf{try\_to\_commit}_T()$. The transaction T locks all the objects it has accessed (they are the objects in $lrs_T \cup lws_T$, line 11). The locking is done according to a canonical order to prevent deadlocks. If it is a read-only transaction (that has read more than one object), it can be committed if its incremental snapshot is still valid, i.e., the values it has read from the shared memory have not yet been overwritten. This is exactly what is captured by the predicate $ConsistencyCheck_T$ (defined at line 10 and used at line 12). If this predicate is true, the transaction appears as if it was atomically executed just before the predicate evaluation. The transaction is then committed. If the predicate is false, there is no way to know if the transaction could be correctly serialized with respect to the committed transactions; it is consequently aborted (line 12).

If the transaction T is write-only (i.e., $lrs_T = \emptyset$, line 12), due to the locks on the objects of lws_T, the transaction T can atomically write their new values into the shared

memory (line 14). Before these writes, T has to update the sequence number of each object X it writes so that the dependency vectors (vector timestamps) have correct values (line 13).

If the transaction T is neither read-only, nor write-only, it can be committed only if all its read and write operations could have been executed atomically. As just seen, the locks ensure that the writes appear as being executed atomically. To check if both reads and the writes of T can appear as being executed atomically, the predicate $ConsistencyCheck_T$ is evaluated, and this evaluation is done after the locks on the objects in $lrs_T \cup lws_T$ have been acquired. If it is evaluated to true, the transaction appears as being executed atomically after the locks have been acquired and consequently the transaction T can be committed. Otherwise it is aborted (line 12).

Let us finally observe that, if a transaction is committed (line 18), the dependency vector of the process p_i has to be updated accordingly (line 17) to take into account the new dependencies created by the newly committed transaction T.

Property 4. If $(abort, 2)$ is returned to a read-only transaction T, the values it has incrementally read define a consistent snapshot, but this snapshot cannot be serialized (with certainty) with respect to the committed transactions.

Property 5. If $(abort, 2)$ is returned to a read/write transaction T, the values it has incrementally read define a consistent snapshot, but this snapshot and the writes into the shared memory cannot appear as being executed atomically.

In the case of the properties 4 and 5, all the read operations issued by the aborted transaction T belong to its read prefix, and this read prefix is consistent with respect to the causal past of T.

Property 6. A write-only transaction cannot be aborted.

Definition 1. *$T1$ and $T2$ are independent if* $(lrs_{T1} \cup lws_{T1}) \cap (lrs_{T2} \cup lws_{T2}) = \emptyset$.

Property 7. Concurrent transactions that are independent can commit independently.

Remark. A simple modification of the previous protocol provides us with the following additional property: a read-only transaction T that reads a single object X is never aborted. T is then only made up of $X.\mathsf{read}_T()$, and this operation is implemented as follows:

if (there is no local copy of X) **then**
allocate local space -denoted $lc(X)$- for a local copy of X;
lock(X); $lc(X) \leftarrow X$; unlock(X)
end if;
return$(lc(X).value)$.

3.3 Properties of the Protocol

Proof. The previous section has stated a few properties whose aim is to give a better intuition of what the algorithms described in Figure 2 do and how they do it. The proof

that they satisfy the virtual consistency condition requires a formal statement of that condition. Due to page limitation this formal statement and the corresponding proof cannot appear in the paper. The reader can consult [19] where are presented a formal definition of the virtual consistency condition and a proof of the algorithms. The committed transactions can be linearized, and the appropriate read prefixes of each aborted transaction are consistent wrt their causal past.

Cost. It is easy to see that the following values are upper bounds on the number of shared memory accesses issued by a transaction:

- $2|lrs_T|$ if T is read-only (lines 02 and 12),
- $2|lws_T|$ if T is write-only (lines 13 and 14), and
- $2|lrs_T| + 2|lwt_T|$ if T is a read/write transaction.

There is the additional cost due to locking/unlocking of base objects (lines 12 and 16). For the objects that are written this cost can be eliminated by placing the lock inside the object and (as in TL2 [8]) aborting a transaction when it accesses an object that is locked.

4 Versatility Dimension of Protocol

4.1 From Virtual World Consistency to Causal Consistency

Causally consistent transactions. The concept of *causal consistency* for read/write objects has been introduced in [1] under the name *causal memory*. It has then been extended to transactions in [22] where only the committed transactions are considered. As for virtual world consistency, we extend here causal consistency to include the appropriate prefixes of the aborted transactions.

Intuitively, given an execution of a set of transactions issued by sequential processes, causal consistency allows each process to see its own "witness sequential execution" as long as these witness sequential executions respect the causal dependencies defined by the "read-from" and "process order" relations.

More precisely, let $\mathcal{C}$ be the set of all the committed transactions that write base objects (whatever the issuing processes). For each process p_i, let $\mathcal{R}_i$ be the set of its committed read-only transactions plus its aborted transactions reduced to their read prefix (as defined previously in the paper). Causal consistency requires that, for each process p_i, there is a "witness sequential execution" involving only the transactions in $\mathcal{C} \cup \mathcal{R}_i$. Let us notice that all these witness sequential executions share the constraint imposed by the "read-from" and "process order" relations as exhibited in $\mathcal{C}$.

Adapting the protocol. The base protocol described in Figure 2 can be adapted very easily (weakened) to implement causal consistency. The single modification consists in adding the statement "**if** $lws_T = \emptyset$ **then** return($commit$) **end if**" just before line 11.

This modification does not alter the protocol for the aborted transactions whose abort is tagged 1 (line 04). As we have seen, the read prefix of such a transaction defines a consistent snapshot of the values previously read. It is now the same for a read-only

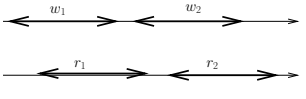
transaction that does not abort at line 04. This is because the lines 11-16 are used to ensure that the consistent snapshot of the values read by the read-only transaction T belongs to the witness sequential execution including all the committed transactions. But, causal consistency does not impose this strong requirement: the values read by a read-only transaction have only to be mutually consistent (and consequently such a transaction can never return $(abort, 2)$ when one is interested in the causal consistency condition).

This shows that causal consistency weakens virtual world consistency by allowing a read-only transaction to commit as long as its snapshot of read values is consistent (as the prefix of an aborted transaction), without requiring that this snapshot be totally ordered with respect to all the committed transactions. The snapshot has only to be consistent with respect to the causal past of the read-only transaction.

4.2 From Atomic Objects to Regular Objects

Regular read/write object. A single writer *regular* read/write object [20] has one writer and any number of readers. Regular objects with multiple writers and multiple readers have been investigated in [25] where three different regularity definitions are presented. Here we consider that the writes appear as being executed sequentially, this sequence complying with their real time order (i.e., if two writes w_1 and w_2 are concurrent they can appear in any order, but if w_1 terminates before w_2 starts, w_1 has to appear as being executed before w_2).

As far as a read operation is concerned we have the following. If no write operation is concurrent with a read operation, that read operation returns the current value kept in the object. Otherwise, the read operation returns any value written by a concurrent write operation or the last value of the object before these concurrent writes. A regular object can exhibit what is called a *new/old inversion*. The figure on the right depicts two write operations w_1 and w_2 and two read operations r_1 and r_2 that are concurrent (r_1 is concurrent with w_1 and w_2, while r_2 is concurrent with w_2 only). According to the definition of regularity, it is possible that r_1 returns the value written by w_2 while r_2 returns the value written by w_1.



An atomic read/write object is a regular read/write object without new/old inversion. This means that an atomic read/write object is such that all its read and write operations appear as if they have been executed sequentially, this total order respecting the real time order of the operations.

Adapting the protocol. If the base objects are regular, we have to prevent new/old inversion so that they appear as if they were atomic. This can be obtained by adding a statement and modifying a predicate. More precisely the following modifications allow us to replace the base atomic read/write objects by weaker regular read/write objects.

– Line 03 is enriched by a test that prevents from reading an old value. That line becomes (the new statement is the **if** statement):

$lrs_T \leftarrow lrs_T \cup \{X\}$;
if $(t\_depend_T[X] > lc(X).depend_T[X])$ **then** $\mathsf{return}(abort, 3)$ **end if**;
$t\_depend_T[X] \leftarrow lc(X).depend[X]$.

– $ConsistencyCheck_T$ becomes $(\forall\ Z \in lrs_T : t\_depend_T[Z] \geq Z.depend[Z])$.

The meaning of the result $(abort, 3)$ returned in the **if ... end if** statement is the following. First, the transaction T has previously read an object (say Y) the value of which depends on the value of X whose sequence number is $sn = t\_depend_T[X]$. The sequence number sn' of X just read by T ($sn' = lc(X).depend_T[X]$)) is such that $sn' < sn$. This witnesses a new/old inversion involving the "early" read of X – issued by some by some T' – that obtained the new value of X to produce the value of Y, and the "late" read of X by T that obtained a previous value of X. While this behavior is impossible when the base objects are atomic, it can happen in concurrency patterns when the base objects $X, Y, \ldots$ are only regular.

Property 8. If the invocation of $X.\mathsf{read}_T()$ by T returns $(abort, 3)$, the abort is due to a new/old inversion.

4.3 When the Base Objects Are Neither Atomic Nor Regular

When the base objects are neither atomic nor regular, there is a very simple way to enrich the protocol of Figure 2 to make it work correctly. In order to make a base object X atomic, it is sufficient to use the lock associated with that object and replace the read of X from the shared memory at line 02 by "lock(X); $lc(X) \leftarrow X$; unlock(X)".

5 Conclusion

This paper has presented a new consistency condition called *virtual world* consistency [18], that is weaker than opacity while keeping its spirit. It has then presented a STM protocol with invisible read operations that implements this condition. This protocol, that is based on vector clocks that capture the causal dependencies among the values of the objects, presents an interesting versatility feature. The suppression of a consistency test provides a protocol satisfying the *causal consistency* condition (that is weaker than virtual world consistency), while the appropriate addition of a simple consistency test allows us to replace the base atomic objects by (weaker) regular objects.

The proposed STM protocol is targeted for applications where the processes share a "reasonable" number of base objects. This is in order to have small size vector clocks. When the application processes share a large number of objects, it is possible to have small size vector clocks by requiring sets of objects to share the same entry of the vector clock as it is done in the "plausible vector clock" approach [27]. In that case, no causal dependency is lost, but additional "false" dependencies can be witnessed by a vector clock. This is due to the fact that several objects share the same entry of the vector clock. The benefit of using such vector clocks the size k of which is bounded and much smaller than m (the number of shared objects) has a price: due to the false additional dependencies, more transactions can be aborted. (Let us remark that the objects that share the same vector clock entry have also to share the same lock.)

Finally, let us notice that both the *virtual world consistency* condition and the associated vector clock-based protocol offer an additional insight on STM systems, that participate in providing a better understanding of their underlying basic principles [3]. Moreover, as the TL2 protocol [8],is based on a scalar clock, it would be be interesting to investigate if the proposed protocol and TL2 could be derived from a more general framework, with scalar clock being the appropriate mechanism for opacity, and vector clock the appropriate mechanism for virtual world consistency. Finally, evaluating the proposed STM system on a realistic benchmark constitutes an interesting direction of a more applied fundamental research.

Acknowledgment

We would like to thank the referees for whose constructive comments helped improve the presentation of the paper.

References

1. Ahamad, M., Neiger, G., Burns, J.E., Kohli, P.: Hutto Ph.W., Causal Memory: Definitions, Implementation, and Programming. Distributed Computing 9(1), 37–49 (1995)
2. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic Snapshots of Shared Memory. Journal of the ACM 40(4), 873–890 (1993)
3. Attiya, H.: Needed: Foundations for Transactional Memory. ACM Sigact News, DC Column 39(1), 59–61 (2008)
4. Attiya, H., Guerraoui, R., Ruppert, E.: Partial Snapshot Objects. In: Proc. 20th ACM Symposium on Parallel Algorithms and Architectures (SPAA 2008), pp. 336–343. ACP Press, ACM Press (2008)
5. Babaoğlu, Ö., Marzullo, K.: Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In: Distributed Systems. Frontier Series, vol. 4, pp. 55–93. ACM Press, New York (1993)
6. Chandy, K.M., Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems. ACM Transactions on Operating Systems 3(1), 63–75 (1985)
7. Cooper, R., Marzullo, K.: Consistent Detection of Global Predicates. In: Proc. ACM/ONR Workshop on Parallel and Distributed Debugging, pp. 167–174. ACM Press, New York (1991)
8. Dice, D., Shalev, O., Shavit, N.: Transactional Locking II. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 194–208. Springer, Heidelberg (2006)
9. Felber, P., Fetzer, C., Guerraoui, R., Harris, T.: Transactions are coming Back, but Are They The Same? ACM Sigact News, DC Column 39(1), 48–58 (2008)
10. Guerraoui, R., Kapałka, M.: On the Correctness of Transactional Memory. In: Proc. 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2008), pp. 175–184. ACM Press, New York (2008)
11. Harris, T., Cristal, A., Unsal, O.S., Ayguade, E., Gagliardi, F., Smith, B., Valero, M.: Transactional Memory: an Overview. IEEE Micro 27(3), 8–29 (2007)
12. Herlihy, M.P., Luchangco, V.: Distributed Computing and the Multicore Revolution. ACM SIGACT News, DC Column 39(1), 62–72 (2008)
13. Herlihy, M.P., Moss, J.E.B.: Transactional Memory: Architectural Support for Lock-free Data Structures. In: Proc. 20th ACM Int'l Symp. on Computer Architecture (ISCA 1993), pp. 289–300 (1993)

14. Herlihy, M.P., Wing, J.M.: Linearizability: a Correctness Condition for Concurrent Objects. ACM Transactions on Programming Languages and Systems 12(3), 463–492 (1990)
15. Imbs, D., Raynal, M.: A Lock-based STM Protocol that Satisfies Opacity and Progressiveness. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 226–245. Springer, Heidelberg (2008)
16. Imbs, D., Raynal, M.: Provable STM Properties: Leveraging Clock and Locks to Favor Commit and Early Abort. In: Garg, V., Wattenhofer, R., Kothapalli, K. (eds.) ICDCN 2009. LNCS, vol. 5408, pp. 67–78. Springer, Heidelberg (2008)
17. Imbs, D., Raynal, M.: Help When Needed, but No More: Efficient Read/Write Partial Snapshots. In: Keidar, I. (ed.) DISC 2009. LNCS, vol. 5805, pp. 142–156. Springer, Heidelberg (2009)
18. Imbs, D., Raynal, M.: On the Consistency Conditions of Transactional Memories. Tech Report #1917, 23 pages, IRISA, Université de Rennes, France (submitted to publication, 2009)
19. Imbs, D., Raynal, M.: A versatile STM protocol with invisible read operations that satisfies the virtual world consistency condition. Tech Report #1923, 20 pages, IRISA, Université de Rennes, France (2009)
20. Lamport, L.: On interprocess communication. Part 1: Models, Part 2: Algorithms. Distributed Computing 1(2), 77–101 (1986)
21. Papadimitriou, Ch.H.: The Serializability of Concurrent Updates. Journal of the ACM 26(4), 631–653 (1979)
22. Raynal, M., Thia-kime, G., Ahamad, M.: From serializable to causal transactions. In: BA. Proc. 20th ACM Symposium on Distributed Computing (PODC 1996), p. 310. ACM Press, New York (1996); Full version: From serializable to causal transactions for collaborative applications. In: Proc. 23th EUROMICRO Conference, pp. 314-321. IEEE Computer Press, Los Alamitos (1997)
23. Riegel, T., Fetzer, C., Felber, P.: Time-based Transactional Memory with Scalable Time Bases. In: Proc. 19th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2007), pp. 221–228. ACM Press, New York (2007)
24. Shavit, N., Touitou, D.: Software Transactional Memory. Distributed Computing 10(2), 99–116 (1997)
25. Shao, C., Pierce, E., Welch, J.: Multi-writer consistency conditions for shared memory objects. In: Fich, F.E. (ed.) DISC 2003. LNCS, vol. 2848, pp. 106–120. Springer, Heidelberg (2003)
26. Schwarz, R., Mattern, F.: Detecting Causal Relationship in Distributed Computations: in Search of the Holy Grail. Distributed Computing 7, 149–174 (1993)
27. Torres-Rojas, F., Ahamad, M.: Plausible Clocks: Constant Size Logical Clocks for Distributed Systems. Distributed Computing 12, 179–195 (1999)

On-Line Maximum Matching in Complete Multipartite Graphs with Implications to the Minimum ADM Problem on a Star Topology

Mordechai Shalom[1,⋆], Prudence W.H. Wong[2,⋆⋆], and Shmuel Zaks[3]

[1] TelHai Academic College, Upper Galilee, 12210, Israel
cmshalom@telhai.ac.il
[2] Department of Computer Science, The University of Liverpool, Liverpool, UK
pwong@liverpool.ac.uk
[3] Department of Computer Science, Technion, Haifa, Israel
zaks@cs.technion.ac.il

Abstract. One of the basic problems in optical networks is assigning wavelengths to (namely, coloring of) a given set of lightpaths so as to minimize the number of ADM switches. In this paper we present a connection between maximum matching in complete multipartite graphs and ADM minimization in star networks. A tight 2/3 competitive ratio for finding a maximum matching implies a tight 10/9 competitive ratio for finding a coloring that minimizes the number of ADMs.

Keywords: Online Matching, Multi-Partite Graphs, Wavelength Assignment, Wavelength Division Multiplexing (WDM), Optical Networks, Add-Drop Multiplexer (ADM).

1 Introduction

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

Given a WDM network $G = (V, E)$ comprising optical nodes and a set of full-duplex lightpaths $P = \{p_1, p_2, ..., p_N\}$ of G, the wavelength assignment (WLA) task is to assign a wavelength to each lightpath p_i. Recent studies in optical networks dealt with the issue of assigning colors to lightpaths, so that every two lightpaths that share an edge get different colors.

When the various parameters comprising the switching mechanism in these networks became clearer the focus of studies shifted, and today many studies concentrate on the total hardware cost. The key point here is that each lightpath uses two Add-Drop Multiplexers (ADMs), one at each endpoint. If two adjacent lightpaths, i.e. lightpaths sharing a common endpoint, are assigned the same

⋆ This research was supported in part by the Israel Science Foundation research grant.
⋆⋆ This research was partly supported by EPSRC Grant EP/E028276/1.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 281–294, 2010.

wavelength, then they can use the same ADM. Because ADMs are designed to be used mainly in ring and path networks in which the degree of a node is at most two, an ADM may be shared by at most two lightpaths. The total cost considered is the total number of ADMs. Lightpaths sharing ADMs in a common endpoint can be considered as concatenated, so that they form longer paths or cycles. These paths/cycles do not use any edge $e \in E$ twice, for otherwise they cannot use the same wavelength which is a necessary condition to share ADMs.

Minimizing the number of ADMs in optical networks is a main research topic in recent studies. The problem was introduced in [GLS98] for the ring topology. An approximation algorithm for the ring topology with approximation ratio of $\frac{3}{2}$ was presented in [CW02], and was improved in [SZ04], [EL04], [EL09] to $\frac{10}{7}+\epsilon$, $\frac{10}{7}$, and $\frac{98}{69}$ respectively. The off-line version of the Minimum ADM problem can be solved optimally for trees [ZCXG03].

The motivation for the on-line problem stems from the need to utilize the cost of use of the optical network. We assume that the switching equipment is installed in the network. Once a lightpath arrives, we need to assign it two ADMs, and our target is to determine which wavelength to assign to it so that we minimize the cost, measured by the total number of ADMs used.

An on-line algorithm with competitive ratio of $\frac{7}{4}$ for any network topology was presented in [SWZ07]. It was shown that this algorithm has an optimal $\frac{7}{4}$ competitive ratio for a ring topology, and an optimal $\frac{3}{2}$ competitive ratio for a path topology.

In this paper we present a connection between matchings in complete multipartite graphs and ADM minimization in star networks. Online bipartite maximum matching problem was introduced in [KVV90] and a $(1-1/e)$-competitive randomized algorithm was proposed , which is optimal [GM08, BM08]. The greedy algorithm is 1/2-competitive and is optimal for deterministic online algorithms. The problem has found applications in other related problems (e.g., [ANR02], [AR05], [AC06]). The problem has also been studied for general weighted graphs in [KP93] where a 1/3-competitive deterministic algorithm is given. In [Sit96], a formula for the cardinality of the maximum matching in complete multipartite graph is given. To the best of our knowledge, there is no work on online maximum matching in multipartite graphs.

We show a tight bound of 2/3 for the competitive ratio of deterministic algorithms for this maximum matching problem, implying a tight 10/9 competitive ratio for the ADM minimization problem.

In Section 2 we describe both problems. The lower bound and upper bound for the competitive ratio are presented in Sections 3 and 4, respectively. We conclude and discuss further research directions in Section 5.

2 Preliminaries

2.1 The ADM Minimization Problem

An instance α of the problem is a pair $\alpha = (G, P)$ where $G = (V, E)$ is an undirected graph and P is a multi-set of simple paths in G. In an on-line instance,

the graph G is known in advance and the set P of paths is given on-line. In this case we denote $P = \{p_1, p_2, ..., p_N\}$ where p_i is the i-th path of the input and $P_i = \{p_j \in P | j \leq i\}$ consists of the first i paths of the input.

A valid chain (resp. cycle) is a path (resp.cycle) formed by the concatenation of distinct paths $p_{i_0}, p_{i_1}, ... \in P$ that do not have an edge in common. A solution S of an instance $\alpha = (G, P)$ is a partition of P into valid chains and cycles.

The cost of a valid chain (resp. cycle) containing k paths is $k + 1$ (resp. k) ADMs. The cost $cost(S)$ of a solution S is the sum of the costs of its valid chains and cycles. The objective is to find a solution S such that $cost(S)$ is minimum.

Let OPT be an optimal off-line algorithm. An online algorithm A to a minimization problem is said to be c-competitive (for $c \geq 1$) if there exists some $b \geq 0$ such that for any input I, $A(I) \leq c \times OPT(I) + b$, where $A(I)$ and $OPT(I)$ denote the cost of the output of A and OPT, respectively, on input I. Similarly, for a maximization problem A is said to be c-competitive (for $c \leq 1$) if there exists some $b \geq 0$ such that for any input I, $A(I) \geq c \times OPT(I) - b$.

2.2 The Star Topology and the d-PARTMM Problem

Let G be a star with d edges, namely $G = (V, E), V = \{0, 1, ..., d\}, E = \{e_1, ..., e_d\}$ and $\forall 1 \leq i \leq d, e_i = (0, i)$. In this case paths in P are of length either 1 or 2. Let $p \in P$ be a path of length 2 with endpoints i and j. For a path p' to be concatenated to p, one of its endpoints should be either i or j. In this case p and p' would share one of the edges e_i, e_j. Therefore paths of length 2 constitute valid chains of size 1 in every solution, and each such path costs 2 ADMs. We therefore assume w.l.o.g. that all the paths are of length 1.

Two paths p, p' of length 1 have always a common endpoint 0. Let i (resp. j) be the other endpoint of p (resp. p'). They can form a valid chain if and only if $i \neq j$. In this case the cost of the valid chain is 3, or in other words 3/2 per path, whereas a path constituting a valid chain costs 2. Therefore our goal is to maximize the number of valid chains of size 2, that is equivalent to find a maximum matching in a complete d-partite graph. This problem will be called the d-PARTMM problem throughout the paper.

The following lemma is proven in [Sit96], we give a sketch of proof for completeness.

Lemma 2.1. *Let $G = (V, E)$ be a complete d-partite graph with N nodes (V is partitioned into parts $V_1, V_2, \cdots, V_d$). G contains a matching of size $\lfloor \frac{N}{2} \rfloor$ if and only if $|V_i| \leq \lceil \frac{N}{2} \rceil$ for every i.*

Sketch of Proof: The 'only if' part is obvious. The 'if' part is proved by induction on N. Assume w.l.o.g. that V_1 and V_2 are the two largest sets among $V_1, V_2, \cdots, V_d$. The induction step stems from the observation that, by matching any node $v \in V_1$ with any node $v' \in V_2$, and deleting these two nodes from G, results in a complete bipartite graph G' of $N - 2$ nodes (and sets $V_1 - \{v\}, V_2 - \{v'\}, V_3, \cdots, V_d$). The proof follows by the inductive hypothesis. □

A d-partite graph having a matching of size $\lfloor \frac{N}{2} \rfloor$ will be called *balanced*.

In an on-line instance, the input consists of d empty parts that are initially empty. The nodes of the graph are revealed one at a time where each node is an element of some part. For each node p_i of the input, an on-line algorithm has to decide to which node $p_j (j < i)$ to match it, or to leave it unmatched. As the graph is a complete d-partite graph p_j is eligible if and only if it is unmatched and it is not in the same part as p_i. Once two nodes are matched, the decision cannot be revoked. An on-line instance will be called *completely balanced* if every prefix of it is balanced.

When $d = 2$, the on-line problem can be solved optimally by the greedy algorithm, which matches to each p_i an unmatched node p_j in the other part of the graph as long as such a p_j exists. In the rest of our work we assume $d \geq 3$.

We conclude this section with the following claim that relates the performances of any solution with respect to these two problems.

Lemma 2.2. *A solution to the d-PARTMM problem is a c-approximation* ($0 \leq c \leq 1$)*, if and only if the corresponding solution to the Minimum ADM problem is a* $\frac{4-c}{3}$*-approximation, with the same additive term.*

Proof. Let MM be the size of a maximum d-partite matching, and let M be the size of a d-partite matching that constitutes c-approximation. There exists a constant $b \geq 0$, such that $M \geq cMM - b$. Let S^* be an optimal solution to the Minimum ADM problem and S be a solution corresponding to the c-approximation.

$$\begin{aligned} cost(S^*) &= 2\,|P| - MM \\ cost(S) &= 2\,|P| - M \leq 2\,|P| - cMM + b = \frac{2\,|P| - cMM}{2\,|P| - MM} cost(S^*) + b \\ &= \frac{2 - c(MM/\,|P|)}{2 - MM/\,|P|} cost(S^*) + b. \end{aligned}$$

As $0 \leq c \leq 1$, the coefficient of $cost(S^*)$ is a non decreasing function of $MM/\,|P|$. Considering that $MM/\,|P| \leq 1/2$ we conclude

$$cost(S) \leq \frac{2 - c/2}{2 - 1/2} cost(S^*) + b = \frac{4 - c}{3} cost(S^*) + b.$$

On the other hand let S be a $c' = \frac{4-c}{3}$ approximation to the Minimum ADM problem and M the size of d-partite matching that it induces. There is a constant b' such that

$$\begin{aligned} cost(S) &\leq c' \cdot cost(S^*) + b' \\ 2\,|P| - M &\leq c'(2\,|P| - MM) + b' \\ M &\geq c'MM + (1 - c')2\,|P| - b' \geq c'MM + (1 - c')4MM - b' \\ &= (4 - 3c')MM - b' = cMM - b'. \end{aligned}$$

□

3 Lower Bound

Lemma 3.1. *For any $c > \frac{2}{3}$ and $d \geq 3$ there is no c-competitive deterministic on-line algorithm for the d-PARTMM problem.*

Proof. Assume, by contradiction that there is a $\left(\frac{2}{3}+\epsilon\right)$-competitive deterministic on-line algorithm ALG for some $\epsilon > 0$. Then there is a constant $b \geq 0$, such that for any online instance I

$$ALG(I) \geq \left(\frac{2}{3}+\epsilon\right) OPT(I) - b$$

where $ALG(I)$ is the size of the matching returned by the algorithm and $OPT(I)$ is the size of the maximum matching.

For any non-negative integer k, consider the instance I containing $2k$ nodes, such that k of them are in V_1 and k of them are in V_2. Obviously $OPT(I) = k$, then

$$ALG(I) \geq \left(\frac{2}{3}+\epsilon\right) k - b = \frac{2}{3}k + \epsilon k - b$$

and ALG leaves $k - ALG(I)$ unmatched nodes at each one of V_1 and V_2. Let I' be the online instance which is obtained by appending to I, $2k$ nodes from part 3. In this phase ALG can not do better than matching the $2(k - ALG(I))$ unmatched nodes to the nodes of V_3. Therefore

$$ALG(I') \leq ALG(I) + 2(k - ALG(I)) = 2k - ALG(I) \leq \frac{4}{3}k - \epsilon k + b. \qquad (1)$$

On the other hand $OPT(I') = 2k$. Then

$$ALG(I') \geq \left(\frac{2}{3}+\epsilon\right) OPT(I') - b = \left(\frac{2}{3}+\epsilon\right) 2k - b \qquad (2)$$

Combining (1) and (2) we get

$$\begin{aligned} \left(\frac{2}{3}+\epsilon\right) 2k - b &\leq \frac{4}{3}k - \epsilon k + b \\ 2\epsilon k - b &\leq -\epsilon k + b \\ k &\leq \tfrac{2b}{3\epsilon} \end{aligned}$$

For any input I with k bigger than the right hand side we reach a contradiction. □

By applying Lemma 2.2 for $c = \frac{2}{3}$, and using Lemma 3.1, we have thus proved:

Theorem 1. *For any $c < \frac{10}{9}$, there is no c-competitive deterministic on-line algorithm for the Minimum ADM problem in star networks.*

4 Upper Bound

4.1 Eliminating Unbalanced Instances

The following lemma shows that the difficult instances of the d-PARTMM problem are the completely balanced instances.

Lemma 4.1. *There is a c-competitive deterministic algorithm for the d-PARTMM problem, if and only if there is a c-competitive deterministic algorithm for it when the instances are restricted to be completely balanced.*

Sketch of Proof: The 'only if' part is immediate. We now show the 'if' part. Let ALG be a c-competitive deterministic on-line algorithm for completely balanced instances, where $0 \leq c \leq 1$. We claim that the following algorithm ALG' is c-competitive for all instances.

ALG'
Initialization:
  $U \leftarrow \emptyset$
On input p_i do: $//I = \{p_1, ..., p_i\}$
  $B \leftarrow I \setminus U$
  If B is completely balanced then
    follow the decision of ALG on input B
  else{// There is exactly one part h with more than $\lfloor \frac{B}{2} \rfloor$ nodes of B
    If p_i is in h then
      leave p_i unmatched
    else{
      choose an arbitrary unmatched node $p_j \in U$ (*)
      match p_i to p_j
    }
    $U \leftarrow U \uplus \{p_i\}$
  }

First note that any instance I which is not completely balanced has prefixes which are not balanced. Each such prefix has one part h which is the "heaviest" part containing more than half of the nodes. As the instance is initially balanced it can be uniquely divided into intervals $B_1, U_1, B_2, U_2, ...$ which are alternately balanced and unbalanced. Each unbalanced interval U_i has a corresponding "heaviest" part h_i.

We describe in detail how the intervals U_i are determined. Consider a step during which the input became unbalanced. This happens necessarily after some odd step $2s_i - 1$ with s_i nodes in h_i. After this step $s_i + 1$ nodes out of $2s_i$ are in h_i. Now consider the first step that the input becomes balanced again. It happens necessarily after some even step $2e_i$ with $2e_i - 2$ nodes in h_i. After this step e_i nodes out of $2e_i - 1$ are in h_i. In this case U_i is the interval from $2s_i$ to $2e_i - 1$ during which the input contained $2e_i - 2s_i$ nodes out of which $e_i - s_i$ are in h_i. As $2e_i - 1$ is the first step that this happens, at any time between these two steps any node not in h_i can be matched to a node in h_i in line (*) of

ALG'. Moreover the nodes of U_i admit a perfect matching where each edge of the matching has an adjacent node in h_i. If we remove the sub-instance U_i from the input we remain with the instance until step $2s_i - 1$ which is balanced. Note that an intervals B_i may possibly be empty.

If the instance terminates with an unbalanced interval, i.e. the instance is unbalanced, then for the last unbalanced interval U_l, more than half of the nodes revealed during U_l, say $x + \delta$ of them, are in h_l where x is the total number of nodes in the other parts. If the instance is balanced let $\delta = 0$. Let $B = B_1, B_2, ...$ and $U = U_1, U_2, ..., U_{l-1}$.

Then ALG' returns $\frac{|U_i|}{2}$ matchings at each interval $U_i, i < l$. And for U_l it returns x matchings. On the other hand ALG "sees" only the sub-instance B that is completely balanced. Therefore for some constant b, it returns at least $c \cdot OPT(B) + b$ matchings. We conclude

$$ALG'(I) = ALG(B) + \sum_{i<l} \frac{|U_i|}{2} + x \geq c \cdot OPT(B) + b + \frac{|U|}{2} + x$$
$$\geq c \left(OPT(B) + \frac{|U|}{2} + x \right) + b.$$

On the other hand

$$OPT(I) \leq \left\lfloor \frac{|I| - \delta}{2} \right\rfloor = \left\lfloor \frac{|B \cup U|}{2} \right\rfloor + x = \left\lfloor \frac{|B|}{2} \right\rfloor + \frac{|U|}{2} + x = OPT(B) + \frac{|U|}{2} + x.$$

We conclude that ALG' is c-competitive. □

4.2 Algorithm MATCHBYRATIO

In this section we present the algorithm $MATCHBYRATIO(\alpha, d)$ for completely balanced instances, where $0 < \alpha \leq 2/3$ and d is the number of parts of the graph. We prove that for any α in this interval the algorithm is α-competitive.

Algorithm MATCHBYRATIO is designed with the lower bound proof in mind. It depends on some constant $0 < \alpha \leq 2/3$ and the number of parts d (we justify the dependency on d in Section 5).

In the preprocessing step, it calculates a value β depending on d. The algorithm attempts to maintain the number of the matchings to be close to α times the optimum (with an additive offset of β). Each time it falls behind this threshold it adds one matching to the output. One node of the matching is the current input node by definition of the problem. We call this node the *matching* node. The other node is chosen arbitrarily from the part having the biggest ratio of unmatched nodes (ratio of number of unmatched nodes to total number of nodes in the part). This node is called the *matched* node.

The following pseudo-code of the algorithm will be helpful in the analysis. The algorithm partitions the nodes into three sets: U is the set of unmatched nodes, MG is the set of matching nodes, and MD is the set of matched nodes.

$MATCHBYRATIO(\alpha, d)$
Initialization:
Calculate β as a function of d // See analysis (Theorem 2)
$U \leftarrow \emptyset$
$MG \leftarrow \emptyset$
$MD \leftarrow \emptyset$

On input p_i do: $//I = \{p_1, ..., p_i\}$
$opt \leftarrow OPT(I)$ $// = \lfloor \frac{i}{2} \rfloor$)
if $|MG| < \lfloor \alpha \cdot opt - \beta \rfloor$ then { // (**)
Let h be the part containing p_i
choose an arbitrary unmatched node p_j from part $h' \neq h$ having a maximal ratio of unmatched nodes
if there is no such node then FAIL (*)
output the edge (p_i, p_j)
$U \leftarrow U \setminus \{p_j\}$
$MD \leftarrow MD \cup \{p_j\}$
$MG \leftarrow MG \cup \{p_i\}$
} else {
$U \leftarrow U \cup \{p_i\}$
}

By the description of the algorithm, it is clearly α-competitive unless it fails in the line marked by (*). It remains to prove that this does not happen if $\alpha \leq 2/3$.

We begin by introducing some notation. U_i (resp. MD_i, MG_i) is the value of the set U (resp. MD, MG) after step i of the algorithm, in other words after it has processed p_i. Let also $M_i \stackrel{def}{=} MG_i \uplus MD_i$ and $T_i \stackrel{def}{=} M_i \uplus U_i$. P is the set of all the input nodes. We denote by $P_1, ..., P_d$ the parts of the multipartite graph, clearly $P = \uplus_{h=1}^{d} P_h$. For any subset Q of P, $X_i(Q) \stackrel{def}{=} X_i \cap Q$ where X stands for any one of U, MD, MG, M or T. Whenever X is the name of a set, its lowercase counterpart x denotes its size. For instance $mg_i(P_h)$ is the number of matching nodes of P_h after input p_i is processed by the algorithm. For a nonempty subset Q of P we define its unmatched ratio as $\rho_i(Q) \stackrel{def}{=} \frac{u_i(Q)}{t_i(Q)}$.

A basic property of the algorithm is that the sizes $u_i, m_i, ...$ do not depend on the input and are functions of i only. However their subdivision, i.e. the sizes $u_i(P_h), m_i(P_h), ...$ depend on the input.

Lemma 4.2. *For all steps $1 \leq i \leq j$, in which the condition in line (**) is true, we have*

$$\alpha i - 2\beta + 2 - 2\alpha \leq m_i < \alpha i - 2\beta + 2 \tag{3}$$

$$(1-\alpha)i + 2\beta - 2 < u_i \leq (1-\alpha)i + 2\beta - 2 + 2\alpha \tag{4}$$

and

$$|m_j - m_i - \alpha(j-i)| < 2\alpha$$
$$|u_j - u_i - (1-\alpha)(j-i)| < 2\alpha.$$

Proof. Omitted. □

We assume by contradiction that the algorithm fails, and then use backward analysis to reach a contradiction. Under the failure assumption we define recursively the following two finite sequences:

i_0 is the step during which the algorithm failed and H_0 is the part containing the input at step i_0. Formally,
$H_0 = P_h$, where $p_{i_0} \in P_h$.

For any $k > 0$:

i_k is the last step before i_{k-1} that a matching is added to the output and none of its nodes are in H_{k-1}. If such a step does not exist then i_k is undefined and i_{k-1} terminates the sequence, otherwise:

$H_k = H_{k-1} \cup P_h$ where $p' \in P_h$ and p' is the matched node at step i_k.

Note that a matching or a failure may occur only at even steps, because $\alpha\lfloor\frac{i}{2}\rfloor$ does not increase at odd steps. Therefore i_k is even for all k. Note also that the length of the sequence is at most $d-1$, because each time a part of the graph is added to H, and at least one part (i.e. the part of the matching node) is left out.

Lemma 4.3. *For any $d \geq 3$ and any $\alpha < 1/2$, $MATCHBYRATIO(\alpha, d)$ does not FAIL if $\beta \geq 1$.*

Proof. Assume by contradiction that the algorithm fails at some step, and let i_0 be the step before the failure. By definition H_0 is the part of the graph containing the input node at this step. All the unmatched nodes should be in H_0, because otherwise the algorithm would pick an unmatched node from $P \setminus H_0$ and construct a matching, thus would not fail. Therefore we have

$$u_{i_0}(H_0) = u_{i_0} > (1-\alpha)i_0 + 2\beta - 2 \geq (1-\alpha)i_0 > i_0/2.$$

On the other hand as the instance is balanced we have $u_{i_0}(H_0) \leq t_{i_0}(H_0) \leq \lceil i_0/2 \rceil = i_0/2$, a contradiction. □

Lemma 4.4. *If $\alpha \leq 2/3$ and $\beta > d$ then*

$$u_{i_k}(H_k) \geq 2(1-\alpha)t_{i_k}(H_k) + 2\beta - 2 - k \geq 2(1-\alpha)t_{i_k}(H_k).$$

Proof. The second inequality follows from $\beta > d$, $k < d$ and $d \geq 3$. We will prove the first inequality by induction on k.

$k = 0$**:** The proof is similar to the proof of Lemma 4.3:

$$u_{i_0}(H_0) = u_{i_0} > (1-\alpha)i_0 + 2\beta - 2 = (1-\alpha)t_{i_0} + 2\beta - 2 \geq 2(1-\alpha)t_{i_0}(H_0) + 2\beta - 2$$

where the last inequality holds because the instance is balanced.

$k > 0$**:** Recall that $i_k < i_{k-1}$ and both even. For readability we denote $m = i_k$ and $n = i_{k-1}$. We will analyze the change in the sizes of the sets U, MG, MD, etc... from step $m+1$ to step n.

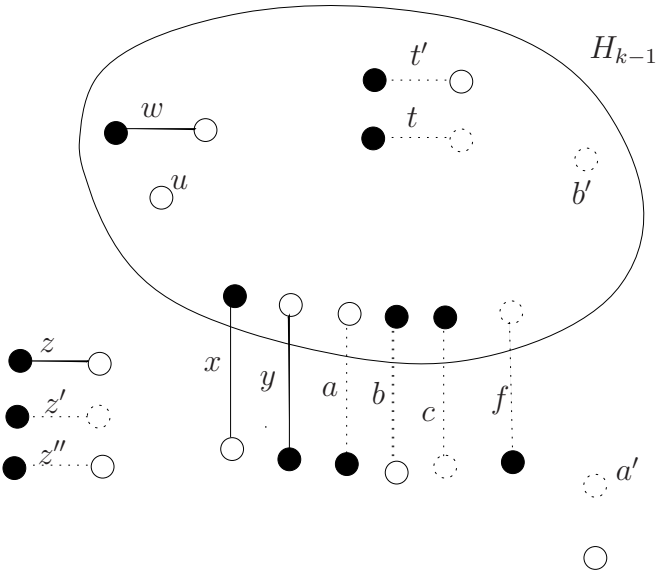


Fig. 1. The scenario discussed in the proof of Lemma 4.4

Consult Figure 1 for the following discussion. Solid edges are matchings that were output until step m and dotted edges are matchings that were output from step $m+1$ to step n. A black node is a matching node, and a white node is a matched node. An unmatched node is drawn with a solid border if it was input until step m, and with a dotted border otherwise. The figure shows all the possibilities of matchings and all the possibilities of unmatched nodes. The letters on the nodes (resp. edges) are the number of such nodes (resp. edges).

First note that m is the last step during which two elements of $\overline{H_{k-1}} = P \setminus H_{k-1}$ are matched to each other. Therefore $z' = z'' = 0$ and the set sizes are as follows:

$$
\begin{aligned}
md_m(H_{k-1}) &= y + w \\
mg_m(H_{k-1}) &= x + w \\
m_m(H_{k-1}) &= x + y + 2w \\
u_m(H_{k-1}) &= a + u + t' \\
t_m(H_{k-1}) &= x + y + 2w + a + u + t' \\
md_n(H_{k-1}) &= y + w + a + f + t + t' \\
mg_n(H_{k-1}) &= x + w + b + c + t + t' \\
m_n(H_{k-1}) &= x + y + 2w + a + b + c + f + 2t + 2t' \\
u_n(H_{k-1}) &= u + b' \\
t_n(H_{k-1}) &= x + y + 2w + a + b + c + f + u + b' + 2t + 2t' \\
mg_n - mg_m &= a + b + c + f + t + t' \\
t_n - t_m &= a' + b' + a + b + 2c + 2f + 2t + t'
\end{aligned}
$$

Claim. $\frac{b'-a-t'}{b+c+f+b'+2t+t'} \leq 2(1-\alpha) + \frac{\alpha}{n-m}$.

Proof.

$$\frac{b'-a-t'}{b+c+f+b'+2t+t'} \leq \frac{b'-a+t}{b+c+f+b'+2t+t'} \leq \frac{a'+b'+c+f+t}{a'+b'+a+b+2c+2f+2t+t'}. \tag{5}$$

The second inequality above holds by the following observation. If $b'-a+t \leq 0$ then the left hand side is non-positive and the right hand side is positive, therefore the inequality holds. Otherwise $b'-a+t > 0$ and the left hand side is a fraction with value of at most 1. Increasing the value of both nominator and denominator by the same value increases the fraction. Note that the right hand side is obtained from the left hand side by adding $a+a'+c+f$ to both nominator and denominator.

On the other hand we have

$$\begin{aligned} &\frac{a'+b'+c+f+t}{a'+b'+a+b+2c+2f+2t+t'} = \frac{(t_n-t_m)-(mg_n-mg_m)}{t_n-t_m} \\ &= \frac{(n-m)-(mg_n-mg_m)}{n-m} < \frac{(n-m)-(\frac{\alpha}{2}(n-m)-\alpha)}{n-m} \\ &= \left(1-\frac{\alpha}{2}\right) + \frac{\alpha}{n-m} \leq 2(1-\alpha) + \frac{\alpha}{n-m}. \end{aligned} \tag{6}$$

Note that the last inequality holds because $\alpha \leq 2/3$. By combining (5) and (6) we get the claim. □

By the inductive assumption we have

$$u_n(H_{k-1}) \geq 2(1-\alpha)t_n(H_{k-1}) + 2\beta - 2 - (k-1),$$

and by the above claim

$$\begin{aligned} b'-a-t' &\leq 2(1-\alpha)(b+c+f+b'+2t+t') + \alpha\frac{(b+c+f+b'+2t+t')}{n-m} \\ &< 2(1-\alpha)(b+c+f+b'+2t+t') + 1. \end{aligned}$$

We combine to get

$$\begin{aligned} u_m(H_{k-1}) &= u_n(H_{k-1}) - (b'-a-t') \\ &> 2(1-\alpha)t_n(H_{k-1}) + 2\beta - 2 - (k-1) \\ &\quad -2(1-\alpha)(b+c+f+b'+2t+t') - 1 \\ &= 2(1-\alpha)t_m(H_{k-1}) + 2\beta - 2 - k > 2(1-\alpha)t_m(H_{k-1}). \end{aligned}$$

Therefore $\rho_m(H_{k-1}) > 2(1-\alpha)$. Now recall that in step m the algorithm matched two nodes, both not from H_{k-1}. Let P_h be the part of the graph containing the matched node. By the behavior of the algorithm this means that the unmatched

ratio of P_h is at least as much as each one of the parts of H_{k-1}, thus at least as much as entire H_{k-1}, therefore $\rho_m(P_h) \geq \rho_m(H_{k-1}) > 2(1-\alpha)$, i.e. $u_m(P_h) > 2(1-\alpha)t_m(P_h)$. Combining with the above, we get:

$$\begin{aligned} u_m(H_k) &= u_m(H_{k-1}) + u_m(P_h) \\ &> 2(1-\alpha)t_m(H_{k-1}) + 2\beta - 2 - k + 2(1-\alpha)t_m(P_h) \\ &= 2(1-\alpha)t_m(H_k) + 2\beta - 2 - k. \end{aligned}$$

□

Lemma 4.5. *For any $d \geq 3$ and $1/2 \leq \alpha \leq 2/3$, $MATCHBYRATIO(\alpha, d)$ does not FAIL if $\beta > \frac{3}{2}d + 3$.*

Proof. If the algorithm fails, the sequences $i_0, i_1, ..., i_l$ and the $H_0, H_1, ..., H_l$ are defined, where $l \leq d-2$. Let $\overline{H_l} = P \setminus H_l$. By the definition of the sequence H and the fact that H_l is the last item of the sequence no matching can have both nodes in $\overline{H_l}$. Such a matching would cause part of $\overline{H_l}$ to be added to H_l, to form H_{l+1}. In other words all the matchings contain at least one node in H_l. Therefore

$$m_{i_l}(\overline{H_l}) \leq m_{i_l}(H_l).$$

α and β satisfy the conditions of Lemma 4.4, by which we have

$$u_{i_l}(H_l) \geq 2(1-\alpha)t_{i_l}(H_l) + \delta = 2(1-\alpha)m_{i_l}(H_l) + 2(1-\alpha)u_{i_l}(H_l) + \delta$$

$$\frac{1}{3}u_{i_l}(H_l) \geq (2\alpha - 1)u_{i_l}(H_l) \geq 2(1-\alpha)m_{i_l}(H_l) + \delta \geq \frac{2}{3}m_{i_l}(H_l) + \delta$$

$$u_{i_l}(H_l) \geq 2m_{i_l}(H_l) + 3\delta.$$

for $\delta = 2\beta - 2 - k$. We conclude

$$u_{i_l} \geq u_{i_l}(H_l) \geq 2m_{i_l}(H_l) + 3\delta \geq m_{i_l}(H_l) + m_{i_l}(\overline{H_l}) + 3\delta = m_{i_l} + 3\delta$$

$$u_{i_l} - m_{i_l} \geq 3\delta.$$

Recalling that $\alpha \geq 1/2$ we get from (3) and (4)

$$u_{i_l} - m_{i_l} \leq (1-2\alpha)i_l + 4\beta + 4\alpha - 4 \leq 4\beta + 4\alpha - 4 \leq 4\beta.$$

Therefore

$$4\beta \geq 3\delta = 6\beta - 6 - 3k$$

$$2\beta \leq 3k + 6 \leq 3d + 6$$

contradicting our assumption. □

Combining Lemma 4.3 and Lemma 4.5 we get

Theorem 2. *For any $d \geq 3$ and any $\alpha \leq 2/3$, $MATCHBYRATIO(\alpha, d)$ does not FAIL if $\beta > \frac{3}{2}d + 3$.*

Corollary 4.1. *$MATCHBYRATIO(2/3, d)$ is a 2/3-competitive algorithm for the $d - PARTMM$ problem, with an additive term of $\frac{3}{2}d + 3$.*

5 Conclusion and Possible Improvements

In this paper, we have shown an interesting connection between maximum matchings in complete multipartite graphs and ADM minimization in star networks. We show a tight 2/3 competitive ratio for finding a maximum matching, implying a tight 10/9 competitive ratio for finding a coloring that minimizes the number of ADMs.

The algorithm used in the upper bound is 2/3-competitive with an additive term β that depends on the number of parts d of the graph, which is supposed to be known in advance. Actually this is the situation for the ADM minimization problem in which the star network (and therefore d) is given in advance. On the other hand our algorithm is usable also when d is not known a priori by a slight modification. We start with the assumption $d = 3$ and increment the value of d each time the first node of some part is revealed, and adjust β accordingly. In this case $\beta = O(d)$ is unbounded and depends on the on-line input. However this does not constitute a problem if d is $o(N)$.

An open question is to improve the competitive ratio by randomized algorithms. It is also interesting to consider other topologies like trees. We believe the result in star networks may be a starting point for the investigation of the more general tree networks.

Another important extension is to consider the ADM minimization problem when grooming is allowed; in graph-theoretic terms, this amounts to coloring the paths so that at most g of them are crossing any edge, and where each ADM can serve up to g paths that come from at most two of its adjacent edges (see [GRS98, ZM03]). Another direction of extension is to the case where more involved switching functions are under consideration.

References

[AC06] Azar, Y., Chaiutin, Y.: Optimal node routing. In: The proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 2006, pp. 596–607 (2006)

[ANR02] Azar, Y., Naor, J(S.), Rom, R.: The competitiveness of on-line assignments. In: The proceedings of the 13th SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, January 2002, pp. 203–210 (2002)

[AR05] Azar, Y., Richter, Y.: Management of multi-queue switches in QoS networks. Algorithmica 43(1-2), 81–96 (2005)

[BM08] Birnbaum, B., Mathieu, C.: On-line bipartite matching made simple. SIGACT News 39(1), 80–87 (2008)

[CW02] Călinescu, G., Wan, P.-J.: Traffic partition in WDM/SONET rings to minimize SONET ADMs. Journal of Combinatorial Optimization 6(4), 425–453 (2002)

[EL04] Epstein, L., Levin, A.: Better bounds for minimizing SONET ADMs. In: Persiano, G., Solis-Oba, R. (eds.) WAOA 2004. LNCS, vol. 3351, pp. 281–294. Springer, Heidelberg (2005)

[EL09] Epstein, L., Levin, A.: Better bounds for minimizing sonet adms. J. Comput. Syst. Sci. 75(2), 122–136 (2009)

[GLS98] Gerstel, O., Lin, P., Sasaki, G.: Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings. In: INFOCOM 1998, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 69–77 (1998)

[GM08] Goel, G., Mehta, A.: Online budgeted matching in random input models with applications to adwords. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, January 2008, pp. 982–991 (2008)

[GRS98] Gerstel, O., Ramaswami, R., Sasaki, G.: Cost effective traffic grooming in WDM rings. In: INFOCOM 1998, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (1998)

[KP93] Kalyanasundaram, B., Pruhs, K.: On-line weighted matching. J. Algorithms 14(3), 478–488 (1993)

[KVV90] Karp, R.M., Vazirani, U.V., Vazirani, V.V.: An optimal algorithm for online bipartite matching (1990)

[Sit96] Sitton, D.: Maximum matchings in complete multipartite graphs. Furman University Electronic Journal of Undergraduate Mathematics 2, 6–16 (1996)

[SWZ07] Shalom, M., Wong, P.W.H., Zaks, S.: Optimal on-line colorings for minimizing the number of ADMs in optical networks. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 435–449. Springer, Heidelberg (2007)

[SZ04] Shalom, M., Zaks, S.: A $10/7 + \epsilon$ approximation scheme for minimizing the number of ADMs in SONET rings. In: First Annual International Conference on Broadband Networks, San-José, California, USA, October 2004, pp. 254–262 (2004)

[ZCXG03] Zhou, F., Chen, G., Xu, Y., Gu, J.: Minimizing ADMs on WDM directed fiber trees. Journal of Computer Science & Technology 18(8), 725–731 (2003)

[ZM03] Zhu, K., Mukherjee, B.: A review of traffic grooming in WDM optical networks: Architecture and challenges. Optical Networks Magazine 4(2), 55–64 (2003)

Loosely-Stabilizing Leader Election in Population Protocol Model

Yuichi Sudo[1], Junya Nakamura[1], Yukiko Yamauchi[2], Fukuhito Ooshita[1], Hirotsugu Kakugawa[1], and Toshimitsu Masuzawa[1]

[1] Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan
{y-sudou,junya-n,f-oosita,kakugawa,masuzawa}@ist.osaka-u.ac.jp
[2] Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, 630-0192, Japan
y-yamauchi@is.naist.jp

Abstract. A self-stabilizing protocol guarantees that starting from an arbitrary initial configuration, a system eventually comes to satisfy its specification and keeps the specification forever. Although self-stabilizing protocols show excellent fault-tolerance against any transient faults (e.g. memory crash), designing self-stabilizing protocols is difficult and, what is worse, might be impossible due to the severe requirements. To circumvent the difficulty and impossibility, we introduce a novel notion of *loose-stabilization*, that relaxes the closure requirement of self-stabilization; starting from an arbitrary configuration, a system comes to satisfy its specification in a relatively short time, and it keeps the specification *for a long time, though not forever*. To show effectiveness and feasibility of this new concept, we present a probabilistic loosely-stabilizing leader election protocol in the Probabilistic Population Protocol (PPP) model of complete networks. Starting from any configuration, the protocol elects a unique leader within $O(nN \log n)$ expected steps and keeps the unique leader for $\Omega(Ne^N)$ expected steps, where n is the network size (not known to the protocol) and N is a known upper bound of n. This result proves that introduction of the loose-stabilization circumvents the already-known impossibility result; the self-stabilizing leader election problem in the PPP model of complete networks cannot be solved without the knowledge of the exact network size.

1 Introduction

A distributed system is a collection of autonomous computational entities (processes) connected by communication links. Fault tolerance of distributed systems has attracted more and more attention since distributed systems are prone to faults. A *self-stabilizing system* [7] has a desirable property that, even when any transient fault (e.g. memory crash) hits the system, it can autonomously recover from the fault. The notion of self-stabilization is described as follows: (i) starting from an arbitrary initial configuration, a system eventually reaches a *safe configuration* (*convergence*), and (ii) once a system reaches a safe configuration,

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 295–308, 2010.

then it keeps its specification forever (*closure*). Although self-stabilizing systems provide excellent fault-tolerance as mentioned above, designing self-stabilizing protocols is difficult and, what is worse, might be impossible due to the severe requirements of self-stabilization.

To circumvent the difficulty and impossibility, many researchers have tried to relax the severe requirement of self-stabilization and proposed a lot of variants. *Probabilistic self-stabilization* [10] guarantees convergence to a safe configuration with probability 1 starting from an arbitrary configuration. *Quasi-stabilization* [11] guarantees convergence to a safe configuration only when all processes in the system start with the program counters of value 0. *Weak-stabilization* [9] guarantees that starting from an arbitrary configuration, there exists an execution that reaches a safe configuration. Devismes et al. [6] investigated the relations among self, probabilistic and weak stabilization. A notable characteristic common to all the above variants is that they relax only the convergence requirement but not the closure requirement of self-stabilization.

In this paper, we adopt *Probabilistic Population Protocol* (PPP) model [2,3] as a distributed system model. The *population protocol* model [1,2,3,4,5,8] is one of the abstract models that represent wireless sensor networks of anonymous mobile sensing devices. In this model, two devices communicate with each other only when they come sufficiently close to each other (we call this event an *interaction*). For example, population protocol model can represent a flock of birds such that each bird is equipped with a sensing device of small transmission range. In such a sensor network, each device can communicate with another device only when the corresponding birds come sufficiently close to each other. The PPP model is a population protocol model with the assumption that any interaction occurs uniformly at random. This assumption is used partly for evaluating time complexity of protocols. We need this assumption because the measure of time is crucial in the concept of loose-stabilization we introduce later.

Self-stabilizing leader election in population protocol model of complete networks is an important problem and has been considered by several papers. Angluin et al. [4] prove that this problem is unsolvable if we can use no information about the network size, in other words, if a protocol must work on the complete networks of finite but any arbitrary size.[1] Cai et al. [5] prove that the exact information of the network size is necessary (and sufficient) to solve the problem. In other words, for any two distinct positive integers n and n', there exists no self-stabilizing leader election protocol that works on both the complete network of size n and the one of size n'. Fischer and Jiang [8] use external entity (a kind of failure detector) to solve the problem with no knowledge of the network size. All of these results can be applied for PPP model. For example, in PPP model of complete networks, a probabilistic stabilizing leader election protocol exists if and only if the protocol knows the exact network size.

[1] They prove this impossibility for a certain kind of class of topology. By this result, the impossibility holds for, for example, complete networks, directed line networks, and connected networks with a certain degree bound and so on.

1.1 Our Contribution

To circumvent difficulty and impossibility in designing self-stabilizing protocols, we introduce a novel notion of *loose-stabilization*, which relaxes the closure requirement of self-stabilization. To the best of our knowledge, this is the first trial to relax the closure requirement and not the convergence requirement. Intuitively, the notion of loose-stabilization is described as follows: (i) starting from an arbitrary configuration, a system reaches a *loosely-safe configuration* within a short time (*convergence*), and (ii) once a system reaches a loosely-safe configuration, then it keeps its specification for a long time (*loose-closure*). In other words, we relax the closure requirement by allowing a system to deviate from its specification even after a loosely-safe configuration but only after a long period satisfying the specification. The requirement of fast convergence is added to guarantee that most of the system running time should satisfy the specification. Actually, the loose-stabilization is practically equivalent to self-stabilization if the specification is kept for a significantly long time (e.g. exponential order with the network size) after the loosely-safe configuration.

Several definitions for the above notion can be formulated, and in this paper, we give a concrete definition of *probabilistic loose-stabilization*, which ensures fast convergence and a long period of closure in terms of *expected time*.

To show effectiveness and feasibility of loose-stabilization, we present a probabilistic loosely-stabilizing leader election protocol in the PPP model of complete networks. The protocol uses the knowledge of an upper bound, say N, of the network size: the protocol works correctly on the networks of any size less than or equals to N. Starting from an arbitrary configuration, the protocol elects a unique leader within $O(nN \log n)$ expected steps, and then, keeps the unique leader for $\Omega(Ne^N)$ expected steps where n is the actual network size. This result discloses an evidence that introduction of the loose-stabilization can circumvent impossibility results on self-stabilization; the self-stabilizing leader election in the PPP model of complete networks cannot be solved even in a probabilistic way without knowledge of the exact network size (as mentioned above). Our protocol uses $O(\log N)$ space per device while prior papers on population protocols usually do not allow each devices to use more than constant space (with respect to n). However, the importance of our protocol is never impaired by this fact because the above impossibility holds even if each device can use infinite space.

2 Preliminaries

In this section, we give the definition of probabilistic population protocol model and define the concept of probabilistic loose-stabilization. We use some definitions in [2],[4].

A *population* consists of a collection of finite state sensing devices called *agents*. Each agent has its own state and updates the state by communication with other agents in pairs, called *interactions*. We represent a population by simple directed graph $G(V, E)$: vertex set $V = \{0, 1, \ldots, n-1\}$ $(n \geq 2)$ represents a set of agents, and edge set $E \subseteq V \times V$ represents a set of possible interactions.

If $(u, v) \in E$, agents u and v can interact with each other in such a way that u serves as an *initiator* and v serves as a *responder*. In this paper, we assume that a population $G(V, E)$ is a complete graph, i.e. $(u, v) \in E$ holds for any distinct agents $u, v \in V$.

A *protocol* $P(Q, Y, O, \delta)$ consists of a finite set of states Q, a finite set of output symbols Y, an output function $O : Q \to Y$, and a transition function $\delta : Q \times Q \to Q \times Q$. The *output of an agent* is determined by O: when the state of an agent is $p \in Q$, the output of the agent is $O(p)$. When an interaction between two agents happens, δ determines the next states of the two agents after the interaction. For agent u with state p and agent v with state q, $\delta(p, q) = (p', q')$ represents that the states of u and v after the interaction (u, v) are p' and q' respectively.

A *configuration* is a mapping $C : V \to Q$ that specifies the states of all agents in a population. The output of a configuration C is defined as a composite function $O \circ C : V \to Y$, denoted by $O(C)$. Let C and D be configurations, and let u and v be distinct agents. We say that C changes to D by an interaction $r = (u, v)$, denoted by $C \xrightarrow{r} D$, if we have $(D(u), D(v)) = \delta(C(u), C(v))$ and $D(w) = C(w)$ for all $w \in V$ except u and v.[2] We denote by $\mathcal{C}_{\mathrm{all}}(P)$ the set of all configurations of P.

An *interaction sequence* $\gamma = (u_0, v_0), (u_1, v_1), \ldots$ is an infinite sequence of interactions. For each $t \geq 0$, we denote u_t and v_t by $\gamma_1(t)$ and $\gamma_2(t)$ respectively, and denote (u_t, v_t) by $\gamma(t)$. We call $\gamma(t)$ *the interaction at time* t *in* γ. We say that agent v *joins in* interaction $\gamma(t)$ when $v \in \{\gamma_1(t), \gamma_2(t)\}$.

Given an interaction sequence γ and an initial configuration C_0, the *execution* $\Xi_P(C_0, \gamma)$ of a protocol P is uniquely defined as $\Xi_P(C_0, \gamma) = C_0, C_1, \ldots$ s.t. $\forall t \geq 0,\ C_t \xrightarrow{\gamma(t)} C_{t+1}$.

A scheduler determines which interaction happens at each time t ($t \geq 0$). In this paper, we consider a uniformly random scheduler: the interaction at each time is chosen at random, independently and uniformly from all possible interactions. We represent the choice of this scheduler by the interaction sequence Γ: each $\Gamma(t)$ is a random variable such that $\Pr(\Gamma(t) = (u, v)) = \frac{1}{|E|}$ for any arbitrary interactions $(u, v) \in E$ and for any integer $t \geq 0$.

2.1 Behavior

In this section, we define *behavior* to describe the specification of a problem. A *trace* T on population $G(V, E)$ is a finite or infinite sequence of assignments from V to Z, where Z is a set of symbols. We call Z the *alphabet* of T. If $Z = Q$ for protocol $P(Q, Y, O, \delta)$ then we say that T is a *configuration trace* of P.[3] Let $T = C_0, C_1, \ldots$ be a finite or infinite configuration trace of P. The *output trace* of T for P is $OT_P(T) = O(C_0), O(C_1), \ldots$.

[2] This definition implies that interactions between two agents happen sequentially, that is, exactly one pair of agents interact at any time.

[3] Note that a configuration trace of $P(Q, Y, O, \delta)$ is also a configuration trace of $P'(Q', Y', O', \delta')$ if $Q = Q'$.

For a finite trace $T = \lambda_0, \lambda_1, \ldots, \lambda_{l-1}$, we define the length of T as $|T| = l$. For an infinite trace T', we define $|T'| = \infty$. Let $T = \lambda_0, \lambda_1, \ldots$ be a finite or infinite trace. The *sub-trace* $T_{x,y}$ $(0 \leq x \leq y \leq |T| - 1)$ [4] is a sequence of assignments $T_{x,y} = \lambda_x, \lambda_{x+1}, \ldots, \lambda_y$. The *prefix* of T, $T_{0,l}$ $(0 \leq l \leq |T| - 1)$ is denoted by $T_{\mathrm{pre}}(l)$.

A *behavior* $B(Z)$ on population $G(V, E)$ is a set of traces on G that have an identical alphabet Z. (We use the notation B if Z is clear from context.) We define a *problem* as a behavior that specifies the set of all legitimate output traces for the problem. Let $B(Y)$ be a behavior and let T be a configuration trace of $P(Q, Y, O, \delta)$. Trace T is legitimate for the problem defined by B iff $OT_P(T) \in B$. We say that a behavior B is *canonical* if $T_{x,y} \in B$ for any trace $T \in B$ and any x, y $(0 \leq x \leq y \leq |T| - 1)$.

Definition 1 (Leader Election Problem). *We denote by le the set of all assignment $\omega : V \to \{F, L\}$ such that for some $v_l \in V$, $\omega(v_l) = L$ and for all $v \neq v_l$, $\omega(v) = F$. The leader election behavior $LE(\{F, L\})$ on population $G(V, E)$ is the set of all traces $T = \omega, \omega, \ldots$ $(1 \leq |T| \leq \infty)$ such that ω belongs to le.*

Informally, *LE* requires that any legitimate execution of a protocol for leader election has one static leader agent with the output symbol L and $n - 1$ non-leader (follower) agents with the output symbol F through its all configuration. Clearly, *LE* is canonical.

2.2 Probabilistic Loose-Stabilization

In this section, we define the notion of *probabilistic loose-stabilization*.

Let $P(Q, Y, O, \delta)$ be a protocol and $B(Y)$ be a canonical behavior. Let $T = D_0, D_1, \ldots$ be a finite or infinite configuration trace of P. If there exists an integer t $(t \geq 0)$ such that $OT_P(T_{\mathrm{pre}}(t)) \in B$ and $OT_P(T_{\mathrm{pre}}(t+1)) \notin B$, the *maintenance trace* $MT_P(T, B)$ is defined by $T_{\mathrm{pre}}(t)$. If such t does not exist, we define $MT_P(T, B)$ as follows: if $OT(T_{\mathrm{pre}}(0)) \in B$ then $MT_P(T, B) = T$, otherwise $MT_P(T, B) = \varepsilon$, where ε is the empty trace $(|\varepsilon| = 0)$. Let C_0 be a configuration of P. We denote $\mathbf{E}[|MT_P(\Xi_P(C_0, \Gamma), B)|]$ by $EMT_P(C_0, B)$. Intuitively, when an execution of P starts from C_0, the execution satisfies the specification defined by B during $EMT_P(C_0, B)$ expected interactions.

Let $\mathcal{C}$ be a set of configurations of P. If there exists an integer t such that $D_i \notin \mathcal{C}$ for all i $(i = 0, 1, \ldots, t)$ and $D_{t+1} \in \mathcal{C}$, the *convergence trace* $CT_P(T, \mathcal{C})$ is defined by $T_{\mathrm{pre}}(t)$. If such t does not exist, we define $CT_P(T, \mathcal{C})$ as follows: if $D_0 \in \mathcal{C}$ then $CT_P(T, \mathcal{C}) = \varepsilon$, otherwise $CT_P(T, \mathcal{C}) = T$. We denote $\mathbf{E}[|CT_P(\Xi_P(C_0, \Gamma), \mathcal{C})|]$ by $ECT_P(C_0, \mathcal{C})$. Intuitively, when an execution of P starts from C_0, the execution reaches a configuration of $\mathcal{C}$ within $ECT_P(C_0, \mathcal{C})$ expected interactions.

Definition 2 (Probabilistic Loose-stabilization). *Let α and β be real numbers. A protocol $P(Q, Y, O, \delta)$ is (α, β)-probabilistic loosely-stabilizing for a*

[4] Note that y can be ∞ if $|T| = \infty$. We interpret $\infty - 1$ as ∞.

canonical behavior $B(Y)$ and a nonempty set of configurations $\mathcal{S}$ if the following equations hold:

$$\max_{C \in \mathcal{C}_{\mathrm{all}}(P)} ECT_P(C, \mathcal{S}) \leq \alpha,$$
$$\min_{C \in \mathcal{S}} EMT_P(C, B) \geq \beta.$$

We say that a configuration C of P is a *β-loosely-safe configuration* for P and B when $EMT_P(C, B) \geq \beta$. Clearly, $\mathcal{S}$ in the above definition consists of β-loosely-safe configurations for P and B.

Intuitively, a (α, β)-probabilistic loosely-stabilizing protocol is quite useful if β is sufficiently large (e.g. exponential order with n) and α is relatively small (e.g. low polynomial order with n).

3 Probabilistic Loosely-Stabilizing Leader Election

3.1 The Proposed Protocol

In this section, we present a leader election protocol $P_{LE}(Q, \{F, L\}, O, \delta)$ working with the knowledge of an upper bound N of the network size n. The protocol has a design parameter s. When s is adequately set depending on N, it is $(O(nN \log n), \Omega(Ne^N))$-probabilistic loosely-stabilizing for behavior LE and a set of configurations S_{half} that we shall define later (Theorem 2).

Each agent has one *leader bit* and a *timer* that takes an integer value in $[0, s]$, i.e. $Q = \{-, l\} \times \{0, 1, \ldots, s\}$. We define the output function O as follows: if the leader bit of an agent is l, then the output of the agent is L, otherwise F. We call an agent with the leader bit l $(-)$ a leader (non-leader, respectively). We describe the transition function δ by pattern rules in Fig. 1. Given any pair of states (p, q), the pair of the next states $\delta(p, q)$ is defined as follows: (i) if (p, q) matches the left side of exactly one rule, $\delta(p, q)$ is determined by the right side of the rule, and (ii) if there are two or more matched rules, $\delta(p, q)$ is determined by the right side of the matched rule with the smallest rule number. The symbol $*$ means "don't care", that is, $*$ matches any value of the timer. Note that this five rules are collectively exhaustive.

If two leaders interact, one remains a leader and the other becomes a non-leader (R1). If a leader and a non-leader interact, the leader bits of the both

	R1.	$((l,*),(l,*))$	$\to$ $((l,s),(-,s))$	
R2.		$((l,*),(-,*))$	$\to$ $((l,s),(-,s))$	
R3.		$((-,*),(l,*))$	$\to$ $((-,s),(l,s))$	
R4.		$((-,0),(-,0))$	$\to$ $((l,s),(-,s))$	
R5.		$((-,i),(-,j))$	$\to$ $((-,f),(-,f))$	$(0 \leq i, j \leq s,\ f = \max(i,j) - 1)$

Fig. 1. the transition function δ of P_{LE}

agents do not change (R2, R3). In every interaction in which one or two leaders join, the timers of both the agents are reset to the full value s (R1, R2, and R3). We call this event *timer reset*. A new leader is created only when two non-leaders with timer value 0 interact (R4). We call this event *timeout*. If two non-leaders interact where either or both agents have non-zero timer, then at least one of the two agents decrements its timer value by 1 (R5). R5 plays another role of *propagating the higher timer value*: intuitively, when two non-leaders interact, the timer of a lower value is set to the other (higher) value (minus 1).

In a configuration containing at least one leader, timeout rarely happens because of frequent occurrences of timer reset and propagations of higher timer value. On the other hand, in a configuration containing no leader, timeout happens in a relatively short time because of no possibility of timer reset. Hence, starting from any configuration, removing leaders by R1 or creating a leader by R4 eventually bring the population to a configuration with exactly one leader. The following two properties hold clearly: (i) once a configuration with one or more leaders is reached, the number of leaders cannot become 0 thereafter, and (ii) once a unique leader is elected, P_{LE} keeps the unique leader until the next timeout happens.

We define $\mathcal{S}_{\mathrm{half}}$ as the set of all configurations in which there exists exactly one leader and the timer value of every agent is greater than or equal to $\frac{s}{2}$. From the above explanation for P_{LE}, one can intuitively observe following two properties: starting from any configuration, the population reaches a configuration in $\mathcal{S}_{\mathrm{half}}$ within a relatively short time (*convergence*), and once a configuration in $\mathcal{S}_{\mathrm{half}}$ is reached, the specification (the unique and static leader) is kept for a extremely long time (*loose-closure*). In the rest of Sect. 3, we show rigorously how fast P_{LE} converges to a loosely-safe configuration, and how long P_{LE} maintains the behavior of leader election after a loosely-safe configuration is reached.

3.2 Epidemic and Virtual Agents

In this section, we introduce the notion of *epidemic* (presented in [3]) and *virtual agents* for the proof in Sect. 3.3.

We define $\mathcal{L}_{\mathrm{one}}$ as the set of all configurations in which there exists exactly one leader in the population. Let C_0 be a configuration in $\mathcal{L}_{\mathrm{one}}$, and let $v_l \in V$ be the unique leader in C_0. Let γ be an interaction sequence. The *epidemic function* $I_{C_0,\gamma}(t)$ $(t = 0, 1, \dots)$ that returns a set of agents is defined as follows: $I_{C_0,\gamma}(0) = \{v_l\}$, and $I_{C_0,\gamma}(t) = I_{C_0,\gamma}(t-1) \cap Add_{C_0,\gamma}(t)$ for any $t \geq 1$ where

$$Add_{C_0,\gamma}(t) = \begin{cases} \{\gamma_1(t-1), \gamma_2(t-1)\} & \text{if } I_{C_0,\gamma}(t-1) \cap \{\gamma_1(t-1), \gamma_2(t-1)\} \\ \emptyset & \text{otherwise .} \end{cases}$$

We say that, if $v \in I_{C_0,\gamma}(t)$, v is *infected* at time t in the epidemic starting from C_0 under γ, otherwise v is *infection-free* at time t in that epidemic. At time 0, only v_l is infected. An infection-free agent becomes infected when it interacts with an infected agent. Once an agent becomes infected, it remains infected thereafter.

In the following, we define the *virtual agent* $VA_{C_0,\gamma}(v)$ of each agent $v \in V$. We assume that all agents eventually become infected, that is, $I_{C_0,\gamma}(t') = V$ holds for some $t' \geq 0$. The virtual agent $VA_{C_0,\gamma}(v)$ is not defined if no such t' exists for C_0 and γ. Let v_l be the unique leader in C_0 and v be any agent other than v_l. The *infected time* $T_{C_0,\gamma}(v)$ of v is an integer $i \geq 0$ that satisfies $v \notin I_{C_0,\gamma}(i)$ and $v \in I_{C_0,\gamma}(i+1)$. The *parent* of v, denoted by $P_{C_0,\gamma}(v)$, is the agent that infects v. It is formally defined as agent u such that $\{u\} = \{\gamma_1(T_{C_0,\gamma}(v)), \gamma_2(T_{C_0,\gamma}(v))\} \setminus \{v\}$. We define agent $P^k_{C_0,\gamma}(v)$ $(k \geq 0)$ as follows: $P^0_{C_0,\gamma}(v) = v$, and $P^k_{C_0,\gamma}(v) = P_{C_0,\gamma}(P^{k-1}_{C_0,\gamma}(v))$ for $k \geq 1$. Intuitively, $P^k_{C_0,\gamma}(v)$ is v's ancestor k generations back. Obviously, there exists an integer $m \geq 0$ such that $P^m_{C_0,\gamma}(v) = v_l$. For each $0 \leq i \leq m$, let w_i be $P^{m-i}_{C_0,\gamma}(v)$. Note that $w_0 = v_l$ and $w_m = v$. The *infecting path* of v is defined as $v_l = w_0 \to w_1 \to \cdots \to w_m = v$. Let t_i $(1 \leq i \leq m)$ be $T_{C_0,\gamma}(w_i)$. The virtual agent $VA_{C_0,\gamma}(v)$ is a virtual entity that migrates from v_l to v through the infecting path of v. This notion is formalized as *the location of the virtual agent* $L_{C_0,\gamma}(v,t)$ $(t \geq 0)$,which is defined as follows:

$$L_{C_0,\gamma}(v,t) = \begin{cases} v_l & (0 \leq t \leq t_1) \\ w_i & (t_i + 1 \leq t \leq t_{i+1},\ 1 \leq i \leq m-1) \\ v & (t \geq t_m + 1 = T_{C_0,\gamma}(v) + 1). \end{cases}$$

For the leader agent v_l, we define $L_{C_0,\gamma}(v_l,t) = v_l$ for any $t \geq 0$.

Let v be an agent in V.[5] For simplicity, we denote the virtual agent $VA_{C_0,\gamma}(v)$ by v' here. We say that the virtual agent v' joins in interaction $\gamma(t)$ if agent $L_{C_0,\gamma}(v,t)$ joins in $\gamma(t)$, and we define indicator variable $VJ_{C_0,\gamma}(v,t)$ for any $t \geq 0$ as follows: if v' joins in $\gamma(t)$, then $VJ_{C_0,\gamma}(v,t) = 1$, otherwise $VJ_{C_0,\gamma}(v,t) = 0$. *The number of virtual interactions* of v, denoted by $VI_{C_0,\gamma}(v,t)$, is defined as $\sum_{i=0}^{t-1} VJ_{C_0,\gamma}(v,i)$. Intuitively, $VI_{C_0,\gamma}(v,t)$ is the number of interactions in which v' joins between time 0 and time $t-1$.

In the rest of this section, we prove two lemmas. Informally, these two lemmas assure that the virtual agent v' brings an large timer value to v with high probability when v' reaches v through the infecting path of v. For state p, we denote the second element (timer) of p by $p.time$.

Lemma 1. *Let C_0 be a configuration in $\mathcal{L}_{\text{one}}$ and let γ be an interaction sequence. Let $\Xi_{P_{LE}}(C_0,\gamma) = C_0, C_1, \ldots$. The following predicate holds for any agent $v \in V$ and any $t \geq 0$:*

$$I_{C_0,\gamma}(t) = V \ \Rightarrow\ C_t(v).time \geq s - VI_{C_0,\gamma}(v,t).$$

Proof Sketch. Assume $I_{C_0,\gamma}(t) = V$. Let v_l be the unique leader in C_0 and t_{first} be the first time at which v_l have interaction, i.e. $t_{\text{first}} = \min\{i \geq 0 \mid v_l \in \{\gamma_1(i), \gamma_2(i)\}\}$. Then, it is easily shown by induction with respect to i that $C_i(L_{C_0,\gamma}(v,i)).time \geq s - VI_{C_0,\gamma}(v,i)$ holds for any integer $i \geq t_{\text{first}} + 1$ (we omit the proof). Since $I_{C_0,\gamma}(t) = V$, $t \geq t_{\text{first}} + 1$ and $v = L_{C_0,\gamma}(v,t)$ clearly hold. Hence, we have $C_t(v).time = C_t(L_{C_0,\gamma}(v,t)).time \geq s - VI_{C_0,\gamma}(v,t)$. □

[5] Note that v can be v_l.

The following lemma probabilistically bounds the number of virtual interactions of each agent by a certain binomial distribution. Recall that random variable Γ is the interaction sequence that represents the choice of uniformly random scheduler.

Lemma 2. *Let C_0 be a configuration in $\mathcal{L}_{\mathrm{one}}$ and let $X(i)$ be a binomial random variable such that $X(i) \sim B(i, \frac{4}{n})$ for integer $i \geq 0$. $\Pr(VI_{C_0,\Gamma}(v,t) \geq j + n - 1 \mid I_{C_0,\Gamma}(t) = V) \leq \Pr(X(t) \geq j)$ holds for any $v \in V$ and any integers $t \geq n$ and $j \geq 0$.*

Proof. Assume $I_{C_0\Gamma}(t) = V$ and let $v_l \in V$ be the unique leader in C_0. We define the *infecting time set IT* as $\bigcup_{v \in V \setminus \{v_l\}} \{T_{C_0,\Gamma}(v)\}$, and the *non-infecting time set NIT* as $\{0, 1, \ldots, t-1\} \setminus IT$. Let v be any agent in V, and let $NVI = \sum_{t' \in NIT} VJ_{C_0,\Gamma}(v,t')$. Since $|IT| = n-1$, the inequality $VI_{C_0,\Gamma}(v,t) \leq NVI + n - 1$ immediately follows. Therefore, it is sufficient for our proof to show $\Pr(NVI \geq j \mid I_{C_0,\Gamma}(t) = V) \leq \Pr(X(t) \geq j)$.

Let t' be any integer in $[0, t-1]$ and let $m = |I_{C_0,\Gamma}(t')|$. If $t' \in NIT$, the interaction $\Gamma(t')$ must be an interaction such that both agents $\Gamma_1(t')$ and $\Gamma_2(t')$ belong to $I_{C_0,\Gamma}(t')$ or the both agents belong to $V \setminus I_{C_0,\Gamma}(t')$. Thus, letting ${}_0C_2 = {}_1C_2 = 0$, we have

$$\begin{aligned}\Pr(VJ_{C_0,\Gamma}(v,t') = 1 \mid I_{C_0,\Gamma}(t) = V \wedge t' \in NIT \wedge L_{C_0,\Gamma}(v,t') \in I_{C_0,\Gamma}(t')) \\ = \frac{m-1}{{}_mC_2 + {}_{n-m}C_2} \ , \\ \Pr(VJ_{C_0,\Gamma}(v,t') = 1 \mid I_{C_0,\Gamma}(t) = V \wedge t' \in NIT \wedge L_{C_0,\Gamma}(v,t') \notin I_{C_0,\Gamma}(t')) \\ = \frac{n-m-1}{{}_mC_2 + {}_{n-m}C_2} \ .\end{aligned}$$

These inequalities lead $\Pr(VJ_{C_0,\Gamma}(v,t') = 1 \mid I_{C_0,\Gamma}(t) = V \wedge t' \in NIT) \leq \frac{4}{n}$ because $\frac{m-1}{{}_mC_2 + {}_{n-m}C_2} \leq \frac{4}{n}$ and $\frac{n-m-1}{{}_mC_2 + {}_{n-m}C_2} \leq \frac{4}{n}$ hold (We omit the proofs of them).

Note that this upper bound $\frac{4}{n}$ of the probability is independent from any interaction at any time other than t'. Hence, for any set S of $t-n+1$ distinct integers in $[0, t-1]$, we have

$$\begin{aligned}\Pr\left(\sum_{t' \in NIT} VJ_{C_0,\Gamma}(v,t') \geq j \mid I_{C_0,\Gamma}(t) = V \ \wedge \ NIT = S\right) \\ \leq \Pr(X(t-n+1) \geq j) \ .\end{aligned}$$

Therefore, following inequality holds and so does the lemma.

$$\begin{aligned}\Pr(NVI \geq j \mid I_{C_0,\Gamma}(t) = V) &= \Pr\left(\sum_{t' \in NIT} VJ_{C_0,\Gamma}(v,t') \geq j \mid I_{C_0,\Gamma}(t) = V\right) \\ &\leq \Pr(X(t-n+1) \geq j) \\ &\leq \Pr(X(t) \geq j) \ .\end{aligned}$$

□

3.3 Analysis and Proofs

Assume that we set design parameter s so that s is multiple of 96 and $s \geq \max(3n, 96(2\ln n + \ln 24))$. In this section, we prove that under this assumption, P_{LE} is $(O(ns\log n), \Omega(se^{s/96}))$-probabilistic loosely-stabilizing for LE and $\mathcal{S}_{\mathrm{half}}$. To claim it, we prove the following two expressions:

$$\max_{C\in\mathcal{C}_{\mathrm{all}}(P_{LE})} ECT_{P_{LE}}(C, \mathcal{S}_{\mathrm{half}}) \in O(ns\log n), \tag{1}$$

$$\min_{C\in\mathcal{S}_{\mathrm{half}}} EMT_{P_{LE}}(C, LE) \in \Omega\left(s\cdot\exp\left(\frac{s}{96}\right)\right). \tag{2}$$

First, we prove (2). In the following, we denote $\mathcal{C}_{\mathrm{all}}(P_{LE})$ by $\mathcal{C}_{\mathrm{all}}$ for simplicity.

Lemma 3. *Equation (2) holds if the following equation holds for any configuration C_0 in $\mathcal{S}_{\mathrm{half}}$:*

$$\Pr\left((\Xi_{P_{LE}}(C_0,\Gamma))_{\mathrm{pre}}\left(\frac{ns}{48}\right)\in LE \ \wedge\ C_{\frac{ns}{48}}\in\mathcal{S}_{\mathrm{half}}\right) \geq 1-2n\cdot\exp\left(-\frac{s}{96}\right), \tag{3}$$

where $\Xi_{P_{LE}}(C_0,\Gamma) = C_0, C_1, \ldots, C_{\frac{ns}{48}}, \ldots$.

Proof. Assume that (3) holds for any configuration in $\mathcal{S}_{\mathrm{half}}$. Then the inequality $EMT_{P_{LE}}(C_0, LE) \geq (1-2n\cdot\exp(-s/96))(\frac{ns}{48} + \min_{C\in\mathcal{S}_{\mathrm{half}}} EMT_{P_{LE}}(C, LE)))$ clearly holds for any configuration $C_0 \in \mathcal{S}_{\mathrm{half}}$. Hence, we have

$$\begin{aligned}&\min_{C\in\mathcal{S}_{\mathrm{half}}} EMT_{P_{LE}}(C, LE)\\ &\geq \left(1-2n\cdot\exp\left(-\frac{s}{96}\right)\right)\left(\frac{ns}{48} + \min_{C\in\mathcal{S}_{\mathrm{half}}} EMT_{P_{LE}}(C, LE)\right).\end{aligned}$$

Solving this inequality gives us (2). □

In the following, we show that (3) holds for any configuration $C_0 \in \mathcal{S}_{\mathrm{half}}$. Firstly, we prove the probability of $C_{\frac{ns}{48}} \in \mathcal{S}_{\mathrm{half}}$ is sufficiently close to 1 (Lemma 4,5 and Corollary 1). Secondly, we prove that the probability of $(\Xi_{P_{LE}}(C_0,\Gamma))_{\mathrm{pre}}(\frac{ns}{48}) \in LE$ is sufficiently close to 1 (Lemma 6 and Corollary 2).

Lemma 4. *Let C_0 be a configuration in $\mathcal{L}_{\mathrm{one}}$. The following inequality holds:*

$$\Pr\left(\max_{v\in V} VI_{C_0,\Gamma}\left(v,\frac{ns}{48}\right)\leq\frac{s}{2}\ \middle|\ I_{C_0,\Gamma}\left(\frac{ns}{48}\right)=V\right)\geq 1-n\cdot\exp\left(-\frac{s}{36}\right). \tag{4}$$

Proof. Applying Chernoff bounds, $\Pr(Y \geq (1+\delta)\mathbf{E}[Y]) \leq \exp(-\delta^2\mathbf{E}[Y]/3)$ holds for any binomial random variable Y and any real number δ $(0 \leq \delta \leq 1)$ [12, (4.2)]. Let X be an binomial variable such that $X \sim B(\frac{ns}{48}, \frac{4}{n})$. It follows from the above inequality that $\Pr(X \geq \frac{s}{6}) = \Pr(X \geq (1+1)\cdot\mathbf{E}[X]) \leq \exp(-s/36)$. Let v be any agent. By Lemma 2 and the assumption $s \geq 3n$, we have

$$\begin{aligned}&\Pr\left(VI_{C_0,\Gamma}\left(v,\frac{ns}{48}\right)\geq\frac{s}{2}\mid I_{C_0,\Gamma}\left(\frac{ns}{48}\right)=V\right)\\ \leq&\Pr\left(VI_{C_0,\Gamma}\left(v,\frac{ns}{48}\right)\geq\frac{s}{6}+n-1\mid I_{C_0,\Gamma}\left(\frac{ns}{48}\right)=V\right)\quad\because\frac{s}{2}\geq\frac{s}{6}+n-1\\ \leq&\Pr\left(X\geq\frac{s}{6}\right)\leq\exp\left(-\frac{s}{36}\right).\end{aligned}$$

We obtain (4) by summing up all above probabilities with respect to $v \in V$. □

Lemma 5. $\Pr(I_{C_0,\Gamma}\left(\frac{ns}{48}\right) = V) \geq 1 - n \cdot \exp\left(-\frac{s}{96}\right)$ *holds for any configuration* C_0 *in* $\mathcal{L}_{\mathrm{one}}$.

Proof. For each k $(2 \leq k \leq n)$, we define $T(k)$ as integer t such that $|I_{C_0,\Gamma}(t-1)| = k-1$ and $|I_{C_0,\Gamma}(t)| = k$, and define $T(1) = 0$. Intuitively, $T(k)$ is the first time at which there exists k infected agents in the population. Let $X_{\mathrm{pre}} = T(\lceil\frac{n+1}{2}\rceil)$ and $X_{\mathrm{post}} = T(n) - T(n - \lceil\frac{n+1}{2}\rceil + 1)$. Angluin et al. found in [3] that $T(k)$ and $T(n) - T(n-k+1)$ have the same probability distribution for any k $(1 \leq k \leq n)$. Hence, so do X_{pre} and X_{post}. And, $X_{\mathrm{pre}} + X_{\mathrm{post}} \geq T(n)$ holds because $\lceil\frac{n+1}{2}\rceil \geq n - \lceil\frac{n+1}{2}\rceil + 1$. We denote $T(n - \lceil\frac{n+1}{2}\rceil + 1)$ by T_{half} and let $X_v = \max(T_{C_0,\Gamma}(v) - T_{\mathrm{half}}, 0)$ for any agent v. Informally, X_v is the number of interactions that occurs between time T_{half} and the time at which agent v becomes infected. Consider the case $v \notin I_{C_0,\Gamma}(T_{\mathrm{half}})$. At any time $t \geq T_{\mathrm{half}}$, at least $n - \lceil\frac{n+1}{2}\rceil + 1$ $(\geq \frac{n}{2})$ agents are infected. Therefore, each interaction at time $t \geq T_{\mathrm{half}}$ infects v with the probability of at least $\frac{1}{{}_nC_2} \cdot \frac{n}{2} \geq \frac{1}{n}$, and hence, we have $\Pr(X_v > \frac{ns}{96}) \leq (1 - \frac{1}{n})^{ns/96} \leq \exp(-\frac{s}{96})$. Since the number of infection-free agent at time T_{half} is at most $\frac{n}{2}$, $\Pr(X_{\mathrm{post}} > \frac{ns}{96}) \leq \Pr(\bigvee_{v\in V}(X_v \geq \frac{ns}{96})) \leq \sum_{v\in V}\Pr(X_v \geq \frac{ns}{96}) \leq \frac{n}{2} \cdot \exp(-\frac{s}{96})$. By the equivalence of the distribution of X_{pre} and X_{post}, we have

$$\begin{aligned}\Pr\left(I_{C_0,\Gamma}\left(\frac{ns}{48}\right) \neq V\right) &= \Pr\left(T(n) > \frac{ns}{48}\right)\\ &\leq \Pr\left(X_{\mathrm{pre}} > \frac{ns}{96}\right) + \Pr\left(X_{\mathrm{post}} > \frac{ns}{96}\right) \leq n \cdot \exp\left(-\frac{s}{96}\right).\end{aligned}$$

□

We define $\mathcal{L}_{\mathrm{half}}$ to be the set of all configurations in which there exists at least one leader and the timer value of every agent is greater than or equal to $\frac{s}{2}$. Note that $\mathcal{S}_{\mathrm{half}} = \mathcal{L}_{\mathrm{half}} \cap \mathcal{L}_{\mathrm{one}}$. The following corollary is directly obtained from Lemmas 1, 4, and 5.

Corollary 1. *Let* C_0 *be a configuration in* $\mathcal{L}_{\mathrm{one}}$ *and let* $\Xi_{P_{LE}}(C_0, \Gamma) = C_0, C_1, \ldots, C_{\frac{ns}{48}}, \ldots$ *. Then,* $\Pr(C_{\frac{ns}{48}} \in \mathcal{L}_{\mathrm{half}}) \geq 1 - n \cdot \exp(-s/36) - n \cdot \exp(-s/96)$ *holds.*

We define $RJ_\gamma(v,t)$ for any $v \in V$ and any $t \geq 0$ as follows: if v joins in $\gamma(t)$, $RJ_\gamma(v,t) = 1$, otherwise $RJ_\gamma(v,t) = 0$. *The number of real interactions* of v is defined by $RI_\gamma(v,t) = \sum_{i=0}^{t-1} RJ_\gamma(v,t)$. Intuitively, $RI_\gamma(v,t)$ is the number of interactions in which v joins between time 0 and time $t-1$.

Lemma 6. $\Pr(\max_{v\in V} RI_\Gamma(v, \frac{ns}{48}) \leq \frac{s}{2}) \geq 1 - n \cdot \exp(-s/4)$ *holds.*

Proof. For any integer $t \geq 0$ and any agent $v \in V$, the probability that v joins in $\Gamma(t)$ is $\frac{2}{n}$. Hence, $RI_\Gamma\left(v, \frac{ns}{48}\right) \sim B(\frac{ns}{48}, \frac{2}{n})$. Applying Chernoff bounds, $\Pr(Y \geq R) \leq 2^{-R}$ holds for any binomial random variable Y and any real number $R \geq 6 \cdot \mathbf{E}[Y]$ [12, (4.3)]. Since $\frac{s}{2} \geq 6\mathbf{E}[RI_\Gamma(v, \frac{ns}{48})]$ and $\ln 2 \geq \frac{1}{2}$, we obtain

$$\Pr\left(\max_{v\in V} RI_\Gamma\left(v,\frac{ns}{48}\right)\geq\frac{s}{2}\right)\leq\sum_{v\in V}\Pr\left(RI_\Gamma\left(v,\frac{ns}{48}\right)\geq\frac{s}{2}\right)$$
$$\leq n\cdot 2^{-s/2}\leq n\cdot\exp\left(-\frac{s\ln 2}{2}\right)\leq n\cdot\exp\left(-\frac{s}{4}\right).$$

□

Corollary 2. $\Pr((\Xi_{P_{LE}}(C_0,\Gamma))_{\mathrm{pre}}(\frac{ns}{48})\in LE)\geq 1-n\cdot\exp(-s/4)$ *holds for any configuration* C_0 *in* $\mathcal{S}_{\mathrm{half}}$.

Proof. Recall that an execution of P_{LE} staring from a configuration in $\mathcal{L}_{\mathrm{one}}$ keeps its unique leader until next timeout happens (Sect. 3.1). Since $C_0\in\mathcal{S}_{\mathrm{half}}$, timeout happens by time $\frac{ns}{48}-1$ only when some agent joins in at least $\frac{s}{2}+1$ interactions between time 0 and time $\frac{ns}{48}-1$. Therefore, the corollary follows from Lemma 6. □

Theorem 1. $\mathcal{S}_{\mathrm{half}}$ *is a set of* $\Omega(se^{s/96})$*-loosely-safe configurations for LE and* P_{LE}*,* i.e. (2) *holds.*

Proof. By the assumption $s\geq 96(2\ln n+\ln 24)$, $s\geq 96$ holds, and then, $\exp(-\frac{s}{4})+\exp(-\frac{s}{36})\leq\exp(-\frac{s}{96})$ holds. Hence, $\exp(-\frac{s}{4})+\exp(-\frac{s}{36})+\exp(-\frac{s}{96})\leq 2\exp(-\frac{s}{96})$ follows. Therefore, (3) holds for any configuration $C_0\in\mathcal{S}_{\mathrm{half}}$ from Corollaries 1 and 2. Hence, we have (2) by Lemma 3. □

Next, we show (1) to complete our proof. We denote by $\mathcal{L}$ the set of all configurations in which there exists at least one leader. The following inequality clearly holds:

$$\max_{C\in\mathcal{C}_{\mathrm{all}}} ECT_{P_{LE}}(C,\mathcal{S}_{\mathrm{half}})$$
$$\leq\max_{C\in\mathcal{C}_{\mathrm{all}}} ECT_{P_{LE}}(C,\mathcal{L})+\max_{C\in\mathcal{L}} ECT_{P_{LE}}(C,\mathcal{L}_{\mathrm{half}})+\max_{C\in\mathcal{L}_{\mathrm{half}}} ECT_{P_{LE}}(C,\mathcal{S}_{\mathrm{half}}).\tag{5}$$

Therefore, it suffices to show that each term in the right side of (5) belongs to $O(ns\log n)$. We can show that the following three lemmas hold, though we omit the proofs of Lemma 8 and Lemma 9 due to the lack of space.

Lemma 7. $\max_{C\in\mathcal{C}_{\mathrm{all}}} ECT_{P_{LE}}(C,\mathcal{L})$ *belongs to* $O(ns\log n)$.

Proof. We define $\nu(C,i)$ $(0\leq i\leq s)$ as the number of agents with timer value i in configuration C, i.e. $\nu(C,i)=|\{v\in V\mid C(v).time=i\}|$. For any integer i,j $(0\leq i\leq s,\ 1\leq j\leq n)$ we denote by $\mathcal{W}_{i,j}$ the set of all configurations in which there exists no leader, the maximum timer value of all agents is i, and $\nu(C,i)=j$ holds.[6] For any set of configurations $\mathcal{X}\in\mathcal{C}_{\mathrm{all}}$, we denote the complement set $\mathcal{C}_{\mathrm{all}}\setminus\mathcal{X}$ by $\overline{\mathcal{X}}$. Note that $\overline{\mathcal{L}}=\bigcup_{i=0}^{s}\bigcup_{j=1}^{n}\mathcal{W}_{i,j}$.

Let $w_{i,j}$ be $\max_{C\in\mathcal{W}_{i,j}} ECT_{P_{LE}}(C,\overline{\mathcal{W}_{i,j}})$. By the definition of P_{LE}, no interaction increments the maximum timer value of all agents as long as there exists no

[6] Note that $\mathcal{W}_{0,j}=\emptyset$ for any integer j $(1\leq j<n)$

leader in the population. Therefore, once an execution of P_{LE} reaches a configurations in $\overline{\mathcal{W}_{i,j}}$ from a configuration in $\mathcal{W}_{i,j}$, the execution cannot reach any configuration in $\mathcal{W}_{i,j}$ thereafter. Hence, the inequality $\max_{C \in \mathcal{C}_{\mathrm{all}}} ECT_{P_{LE}}(C, \mathcal{L}) \leq w_{0,n} + \sum_{i=1}^{s} \sum_{j=1}^{n} w_{i,j}$ holds.

Let i, j be integers such that $1 \leq i \leq s$, $1 \leq j \leq n$. When an interaction involving an agent with timer value i happens, a configuration C in $\mathcal{W}_{i,j}$ changes to a configuration in $\overline{\mathcal{W}_{i,j}}$. Hence, by one interaction, a configuration in $\mathcal{W}_{i,j}$ changes to a configuration in $\overline{\mathcal{W}_{i,j}}$ with the probability of at least $\frac{j(j-1)+2j(n-j)}{n(n-1)} = \frac{j(2n-j-1)}{n(n-1)}$, from which $w_{i,j} \leq \frac{n(n-1)}{j(2n-j-1)}$ follows. Since $\frac{a-1}{b-1} \leq \frac{a}{b}$ holds for any integer a, b $(1 \leq a \leq b)$, we have

$$w_{i,j} \leq \frac{n(n-1)}{j(2n-j-1)} \leq \frac{n^2}{j(2n-j)} = 1 + \frac{(n-j)^2}{j(2n-j)} \leq 1 + \frac{n-j}{j} = \frac{n}{j}.$$

Clearly, $w_{0,n}$ is 1 with the probability 1. Therefore, we obtain

$$\max_{C \in C_{\mathrm{all}}} ECT_{P_{LE}}(C, \mathcal{L}) \leq w_{0,n} + \sum_{i=1}^{s} \sum_{j=1}^{n} w_{i,j} \leq 1 + ns \cdot H(n) \in O(ns \log n) \ ,$$

where H is the harmonic function. □

Lemma 8. $\max_{C \in \mathcal{L}} ECT_{P_{LE}}(C, \mathcal{L}_{\mathrm{half}})$ *belongs to* $O(ns)$.

Lemma 9. $\max_{C \in \mathcal{L}_{\mathrm{half}}} ECT_{P_{LE}}(C, \mathcal{S}_{\mathrm{half}})$ *belongs to* $O(ns)$.

Thus, we obtain (1) from Lemmas 7, 8, 9 and (5). The following theorem is directly derived from Thorem 1 and (1).

Theorem 2. P_{LE} *is* $(O(ns \log n), \Omega(se^{s/96}))$*-probabilistic loosely-stabilizing for behavior* LE *and* $\mathcal{S}_{\mathrm{half}}$ *if* $s \geq \max(3n, 96(2 \ln n + \ln 24))$ *holds.*

Recall that P_{LE} knows an upper bound N of n. When we set s to be $\max(96N, 96(2 \ln N + \ln 24))$, P_{LE} realize $(O(nN \log n), \Omega(Ne^N))$-probabilistic loose-stabilization for behavior LE and $\mathcal{S}_{\mathrm{half}}$. That is, P_{LE} realizes fast convergence to a loosely-safe configuration (low polynomial order time) and extremely long maintenance of its specification (exponential order time).

4 Conclusion

In this paper, we introduced a novel concept of loose-stabilization and presented a probabilistic loosely-stabilizing leader election protocol in the PPP model of complete networks. This protocol assumes that each device (agent) knows an upper bound of the network size. Starting from an arbitrary configuration, the protocol reaches a loosely-safe configuration within $O(nN \log n)$ expected steps, and then, it keeps a unique leader for $\Omega(Ne^N)$ expected steps, where n is the actual network size and N is a known upper bound of n. This protocol has practical significance from the following reason: the protocol can be practically

considered to attain self-stabilization because of exponentially long time of keeping a unique leader while the self-stabilizing leader election in the PPP model of complete networks is impossible without the knowledge of the exact network size [5].

Our future work is to apply the notion of loose-stabilization to other problems that are known unsolvable or too costly in a self-stabilizing fashion.

Acknowledgements. This work is supported in part by Global COE Program of MEXT, Grant-in-Aid for Scientific Research ((B)17300020, (B)19300017, (B) 20300012) of JSPS, Grant-in-Aid for Young Scientists ((B)18700059) of JSPS, and the Kayamori Foundation of Informational Science Advancement.

References

1. Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 63–74. Springer, Heidelberg (2005)
2. Angluin, D.: Computation in networks of passively mobile finite-state sensors. Distributed Computing 18(4), 235–253 (2006)
3. Angluin, D., Aspnes, J., Eisenstat, D.: Fast Computation by Population Protocols with a Leader. In: Proceedings of Distributed Computing, 20th International Symposium, pp. 61–75 (2006)
4. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing Population Protocols. In: Proceedings of Principles of Distributed Systems, pp. 103–117 (2006)
5. Cai, S., Izumi, T., Wada, K.: Space Complexity of Self-Stabilizing Leader Election in Passively-Mobile Anonymous Agents (to be submitted)
6. Devismes, S., Tixeuil, S., Yamashita, M.: Weak vs. self vs. probabilistic stabilization. In: Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS 2008), pp. 681–688 (2008)
7. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM 17(11), 643–644 (1974)
8. Fischer, M.J., Jiang, H.: Self-stabilizing leader election in networks of finite-state anonymous agents. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 395–409. Springer, Heidelberg (2006)
9. Gouda, M.G.: The Theory of Weak Stabilization. In: Datta, A.K., Herman, T. (eds.) WSS 2001. LNCS, vol. 2194, pp. 114–123. Springer, Heidelberg (2001)
10. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self-stabilizing mutual exclusion. In: Proceedings of the ninth annual ACM symposium on Principles of distributed computing, pp. 119–131. ACM Press, New York (1990)
11. Lin, J.C., Huang, T.C., Yang, C.Z., Mou, N.: Quasi-self-stabilization of a distributed system assuming read/write atomicity. Computers and Mathematics with Applications 57(2), 184–194 (2009)
12. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)

Convergence of Mobile Robots with Uniformly-Inaccurate Sensors

Kenta Yamamoto, Taisuke Izumi, Yoshiaki Katayama, Nobuhiro Inuzuka, and Koichi Wada

Nagoya Institute of Technology, japan
kenken@phaser.elcom.nitech.ac.jp,
{t-izumi,katayama,inuzuka,wada}@nitech.ac.jp

Abstract. We consider the convergence problem of autonomous mobile robots with inaccurate sensors, which may return the erroneous location of other robots. In this paper, we newly introduce a uniform error model, which is a restricted variant of the original observation-error model proposed by Cohen and Peleg [4]. The degree of an observation error is characterized by distance errors and angle errors. While the original model (non-uniform model) allows that two or more points can have different error degrees, the uniform error model assumes that the same amount of error degree is incurred to all observed points in a single observation. The main focus of our study is to reveal how much such uniformity expands the feasibility of the convergence. In the non-uniform error model, it has been shown that no algorithm can achieve the convergence if the maximum error angle is more than or equal to $\pi/3$. This paper shows that the convergence problem is solvable under the uniform error if the maximum error angle is less than $\pi/2$. We also prove that there is no convergence algorithm for the maximum error angle more than or equal to $\pi/2$ even in the uniform error model, which implies the optimality of our algorithm in the sense of angle errors.

Keywords: Convergence problem, Observation error, Uniform-error model, smallest enclosing circle.

1 Introduction

1.1 Background

In recent years, cooperations among a large number of autonomous mobile robots have received much attention. In particular, the algorithmic issues of autonomous mobile robots are actively studied in the literature of the distributed computing. In most of algorithmic studies about autonomous mobile robots, a robot is modeled as a point in a plane, and its capability is quite limited: It is usually assumed that robots are oblivious (no memory to record past situations) and anonymous (no IDs to distinguish two robots). Furthermore, they have no explicit direct means of communication. Typically, the communication between two robots is

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 309–322, 2010.

done in the implicit way that each robot observes the environment, which includes the position of other robots in terms of observer's local coordinate system. A theoretical interest of autonomous mobile robots is to reveal what kinds of coordination tasks can be accomplished by exchanging only such positional or geographic information.

Gathering and convergence problems are popular and fundamental coordination tasks for autonomous mobile robots. In short, given a set of robots with arbitrary initial locations, gathering must make all robots meet in finite time at a point that is not predefined. The convergence problem is a weaker variant of the gathering problem. It requires the distance between any two robots converges to zero (i.e., for every $\epsilon > 0$, there exists a time t_ϵ after which any two robots have a distance within ϵ). Both problems have been actively studied before, and a number of possibility/impossibility results under different assumptions are shown [10,6,9,7,8,5,2]. Especially, it is known that the difference of observation capability is strongly related to the solvability of those problems. The gathering problem is first discussed in [10], which proves that it is impossible to achieve gathering of two oblivious autonomous robots that have no common sense of orientation under the semi-synchronous model. This result is expanded to the general number of robots by Prencipe [8]. These impossibility results are one of reasons to make us focus on the convergence problem. The convergence problem is also considered in several papers [4,3,1]. Since, as we mentioned, the convergence problem is weaker than the gathering, most of those studies assume weaker models in the sense of observation capability. Recently, as such a weaker model, Cohen and Peleg introduced the robot model where each robot suffers observation errors. If a robot A observes another robot B, A may see B at the position which is slightly different from the actual location of B. More precisely, if B is located at $(r\cos\phi, r\sin\phi)$ on A's coordinate system, an observation by A may return the coordinate $(r(1+\epsilon)\cos(\phi+\theta), r(1+\epsilon)\sin(\phi+\theta))$ as the B's location (namely, ϵ and θ represent the error ratio about distance and direction respectively). For both of ϵ and θ, their absolute bounds ϵ_0 and θ_0 are assumed. They show that if the maximum angle error θ_0 can be greater than $\pi/3$, it is impossible to achieve to convergence, and propose a convergence algorithm for any maximum distance error ϵ_0 and maximum angle error θ_0 satisfying $0.2 > \sqrt{2(1-\epsilon_0)(1-\cos\theta_0+\epsilon_0^2)}$.

1.2 Our Results

This paper also considers the convergence problem under a similar inaccurate sensor model. The main focus of our study is the uniformity of observation errors: In the original model, the error ratio can be different for each robot. For example, if a robot A observes two other robots B and C, the returning coordinates of B and C can include different amounts of errors. The uniformity of observation error assumes that all coordinates returned by one observation include the same amount of error (but two distinct observations can have the different error ratio even if they are performed by a same robot.) Our interest is to answer the question how the uniformity assumption enhances the capability of robots in

Table 1. Summary and comparison of our results

Model	Maximum distance error ϵ_0	Maximum angle error θ_0	Possibility of convergence
Non-uniform error model ([4])	Any ϵ_0	$\theta_0 \geq \pi/3$	No
	$0.2 > \sqrt{2(1-\epsilon_0)(1-\cos\theta_0+\epsilon_0^2)}$		Yes
	Other		Open
Uniform error model (This paper)	$0 \leq \epsilon_0 < 1$	$\theta_0 < \pi/2$	Yes(Section 4)
	$\epsilon_0 \geq 1$	$\theta_0 < \pi/2$	Open
	Any ϵ_0	$\theta_0 \geq \pi/2$	No(Section 3)

respect to task solvability. Interestingly, we can show that the assumption relaxes the bound on the error ratio for which the convergence task can be solved. More precisely, assuming uniform error ratio to inaccurate sensor models, we can solve the convergence problem if the maximum distance error ratio is less than one, and the maximum angle error is less than $\pi/2$. We present the summary of our result, which includes the comparison to the previous paper, in Table 1.

1.3 Organization

The following is the organization of this paper. In Section 2 we define the robot model and the uniform error model. In Section 3 we present the impossibility result that robots cannot converge when maximum angle error is more than or equal to $\pi/2$. In Section 4 we present a convergence algorithm for the uniform error model with the maximum distance error ratio less than one and the maximum angle error less than $\pi/2$, and prove its correctness.

2 The Robot Model

The robot model of this paper is an extension from that proposed by Suzuki and Yamashita [10] such that each robot suffers observation errors. The following is a concise outline of our model.

- Each robot is a point without volume that moves freely in 2-D space.
- Robots are anonymous, that is, each robot cannot be distinguished from others by ids, their physical appearances, and so on.
- Robots are oblivious. That is, they cannot remember the history of their executions.
- We only consider uniform algorithms. That is, all robots execute the same algorithm.
- Robots have no direct communication device. Each robot observes a configuration of all robots by its own local coordinate system.
- As the timing model, we adopt the *semi-synchronous model*. At each time unit, a subset of all robots (determined by the scheduler) performs movement synchronously.

– An observation of a robot is inaccurate. An observation result may include wrong locations of robots. An observation error is characterized by distance errors and angle errors. We assume the maximum distance error ϵ_0 the maximum angle error θ_0. All robots know the value of ϵ_0 and θ_0 as information about the accuracy of their observations. We also assume the uniformity of observation errors, which implies all observed robots necessarily have the same amount of distance/angle error in a single observation.

In the following subsections, we present the details of our model.

2.1 The System Model

The system consists of n robots $S = \{s_0, s_1, \cdots, s_{n-1}\}$. Each robot is modeled as a point on the two-dimensional Euclidean plane, and works based on discrete time $0, 1, 2 \cdots$. Each robot has its own local coordinate system whose origin is the current position of the robot. A location of all other robots in an observation result is called *local coordinates*. In contrast, to define the positions of robots consistently, we introduce the global coordinate system on the plane. The coordinate of each robot on the global coordinate system is called *global coordinate*. Notice that the global coordinate system is introduced only for ease of explanations, and thus each robot cannot be aware of them. In what follows, any coordinate is represented by two-dimensional vectors, which is described by bold-faced characters.

Each robot is either *active* or *inactive* at each time. An active robot first observes the locations of all other robots, and computes a destination from the observation result. Since each robot is oblivious and uniform, the destination is computed only from the observation result by a common algorithm. In this sense, an algorithm is formally defined as a deterministic function f that maps a set of coordinates (i.e. the positions of all robots) to a coordinate (i,e, the destination). After the computation, the robot moves toward the computed destination on the local coordinate system. We assume that it is guaranteed that any movement is necessarily completed within one time unit. That is, if a robot is active at t, its location at $t + 1$ is the destination computed at t. The set of active robots at each time is determined by the scheduler. Throughout this paper, we assume *fair* scheduler. It ensures that at least one robot is activated in each time and each robot is activated infinitely often.

Each robot observes the locations of all robots in terms of its local coordinate system. There is no assumption about the direction and unit scale of local coordinate systems. That is, each local coordinate system can have a different direction and unit distance. In this paper, it is assumed that each robot suffers *observation error*, which allows the locations of robots observed by another robot to be different from their actual locations. The detail of observation-error models is explained in the following subsection.

2.2 Observation Error

The observation error is characterized by maximum distance error ϵ_0 and maximum angle error θ_0. We explain the influence that these error factors give by

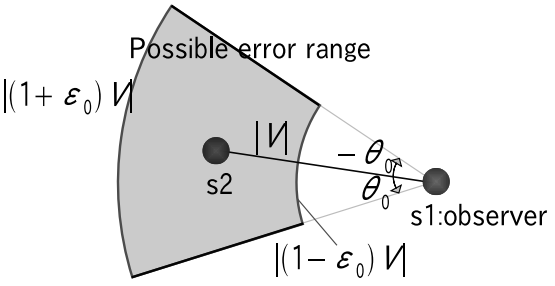


Fig. 1. Observation error

showing an example: Consider a situation where a robot s_1 observes s_2. Let $\mathbf{V}$ be the vector representing the location of a robot s_2 in terms of s_1's coordination system, and $\mathbf{v}$ be the vector representing the location of s_2 in s_1's observation result. (See Fig.1.) Then each error factor is explained as follows:

Distance error
: Any observed distance is affected by at most $\pm\epsilon_0$ fraction of the actual distance. That is, the observation result satisfies $|\mathbf{V}|(1-\epsilon_0) < |\mathbf{v}| < |\mathbf{V}|(1+\epsilon_0)$.

Angle error
: Any observed angle has an additive error within $\pm\theta_0$. That is, letting angle formed by $\mathbf{v}$ and $\mathbf{V}$ be θ, $\cos\theta \geq \cos\theta_0$ is satisfied.

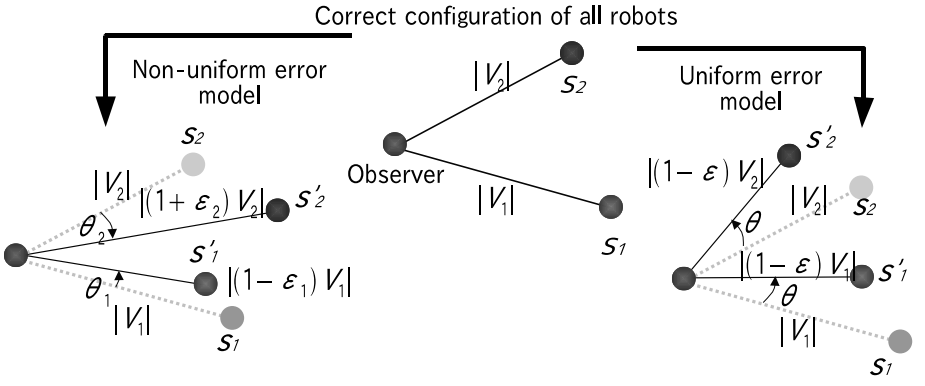
2.3 Uniformity of Observation Error

In this subsection, we define the non-uniform error model and the uniform error model.

Non-uniform error model
: If a robot observes other two or more robots, the observation result can involve different distance/angle errors for each observed robot. An example is shown in Fig.2.a. In this example, robot s_0 observes all other robots, and the observation error occurs differently for two robots s_1 and s_2 : Distance error ratio ϵ_1 and angle error θ_1 is associated with robot s_1 and ϵ_2 and θ_2 with robot s_2.

Uniform error model
: In a single observation, the same observation error is associated with all observed robots (see Fig.2.b), but it is allowed that two different observations have different observation errors.

Notice that even in the uniform-error model, it is possible that two observations by one robot at different timings, and two observations by two robots at the same timing, can have different observation errors.



a. Non-uniform error model

b. Uniform error model

Fig. 2. An example of observations in two each error model

3 Impossibility Result

This section provides the impossibility of the convergence when $\theta_0 \geq \pi/2$.

Theorem 1. *For any number of robots, if $\theta_0 \geq \pi/2$, no algorithm can achieve the convergence.*

Proof. Suppose for contradiction an algorithm A that achieves the convergence for n robots ($n \geq 2$) and $\theta_0 \geq \pi/2$. We start the proof from the initial configuration where all robots are evenly located on a circle. The y-axis of their local coordinate systems are directed to the center of the circle. (See Fig.3). The proof idea is that to find an execution where all robots move to the outside of the circle and form an evenly-located circle again after the movement. Then, by repeating the same execution, the diameter of the circle grows infinitely, which implies the impossibility of convergence.

To construct the desired execution, we first consider the execution where only one robot is activated with no-error observation. Let s_i be the activated robot. If s_i moves to the outside of the circle, we obtain the desired execution by activating all robots simultaneously because, by symmetricity of the configuration, all robots symmetrically move. This implies that they form an evenly-located circle after the movement. On the other hand, If s_i moves to the inside of the circle, we consider an execution where all robots are simultaneously activated but equally suffer angle observation error $\pi/2$. Then, the observation result of each robot is rotated by $\pi/2$, and thus the destination point is also rotated by $\pi/2$. As a consequence, in this execution, all robots symmetrically move toward the outside of the circle. □

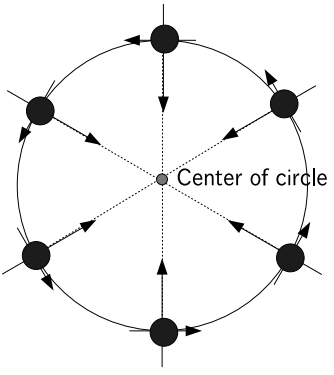


Fig. 3. Initial configuration

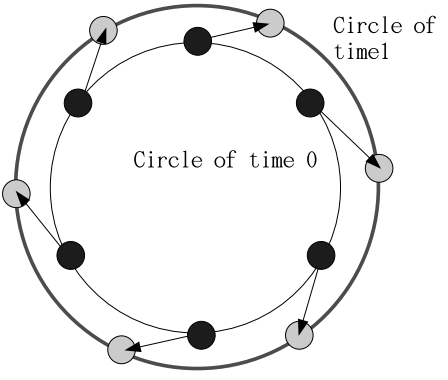


Fig. 4. Symmetric outside movement by robots

4 Convergence Algorithm

4.1 Outline of Algorithm

In this section, we show that a set of robots executing Algorithm *Conv-SEC* converge under the assumption of the uniform error model with $\theta_0 < \pi/2$ and $0 \leq \epsilon_0 < 1$. The pseudo-code of Algorithm *Conv-SEC* shown in the Figure 5. The key idea of the algorithm make robots move toward the center of the smallest-enclosing circle (SEC), which is the minimum-diameter circle containing all positions of robots. At each time, each active robot computes the center of SEC from the observation result (note that for any set of points, its smallest enclosing circle is uniquely determined and it can be computed in polynomial time). Then, if a robot stays on the boundary of SEC, it moves toward the center of SEC with distance $(d \cos\theta_0/(1+\epsilon_0))$, where d is the observed distance between the robot and the center. Notice that the computed center is not equal to the actual center: Because of the observation error, the robot does not move toward the actual center. Then, long-distance movement, that is length d, may cause the robot to go out of the actual SEC. (See Fig.6.) The movement with length $d \cos\theta_0/(1+\epsilon_0)$ ensures robots do not go out of the actual SEC. (See, Fig.7.)

Code for Robot s_i:
1: Observe the locations of all other robots
2: Compute the center of SEC from the observation result
3: **if** s_i is on the computed SEC **then**
4: move toward the center of SEC by distance $d \cos\theta_0/(1+\epsilon_0)$
(d is the distance between s_i and the computed center)
5: **endif**

Fig. 5. Algorithm *Conv-SEC*

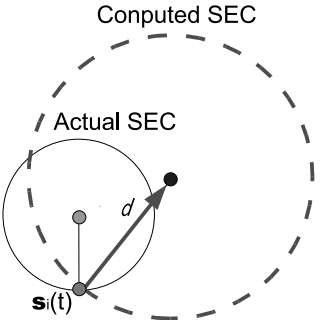


Fig. 6. The movement with length d

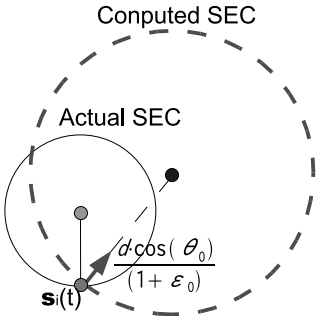


Fig. 7. The movement with length $d\cos\theta_0/(1+\epsilon_0)$

4.2 Correctness Proof

For two points $\mathbf{p}$ and $\mathbf{q}$, let $dis(\mathbf{p},\mathbf{q})$ be the distance between them in terms of the global coordinate system. (i.e., $dis(\mathbf{p},\mathbf{q}) = |\mathbf{p}-\mathbf{q}|$.) The global coordinate of robot s_i at time t is denoted by $\mathbf{s_i}(\mathrm{t})$. The actual center of SEC at time t is denoted by $\mathbf{C}(\mathrm{t})$. Letting t be a time when a robot s_i is active, $\mathbf{c_i}(\mathrm{t})$ denotes the global coordinate of the center of SEC computed by robot s_i at t. For simplicity, we also introduce the following notations:

- $d_i(\mathrm{t}) = dis(\mathbf{c_i}(\mathrm{t}), \mathbf{s_i}(\mathrm{t}))$
- $D_i(\mathrm{t}) = dis(\mathbf{C}(\mathrm{t}), \mathbf{s_i}(\mathrm{t}))$
- $D'_i(\mathrm{t}) = dis(\mathbf{C}(\mathrm{t}), \mathbf{s_i}(\mathrm{t}+1))$

The displacement of the center of actual SEC during $[t, t+1]$ (in terms of the global coordinate system) is denoted by $\Delta(\mathrm{t})$, i.e., $\Delta(\mathrm{t}) = dis(\mathbf{C}(\mathrm{t}), \mathbf{C}(\mathrm{t}+1))$. The radius of SEC at time t is denoted by $\mathbf{R}(\mathrm{t})$. (See, Fig.8.)

It should be noted that in the uniform error model, any observation result is a homothetic transformation of the actual robot locations, whose magnification is between $(1-\epsilon_0)$ and $(1+\epsilon_0)$. This implies that we can obtain the following corollary.

Proposition 1. $(1-\epsilon_0)\, D_i(\mathrm{t}) \leq d_i(\mathrm{t}) \leq (1+\epsilon_0)\, D_i(\mathrm{t})$.

By the nature of the algorithm, it is clear that the diameter of SEC is non-increasing. However, it is not so trivial to prove that the distance certainly converges to zero. The difficulty of the proof is the movement of the center of SEC, which prevents the monotonic decrease of the distance between the center and each robot. This fact implies the necessity of a little more complicated argument: The key idea of our proof is to show that any movement necessarily decreases either the diameter of SEC, or the sum of the distances between the center and robots.

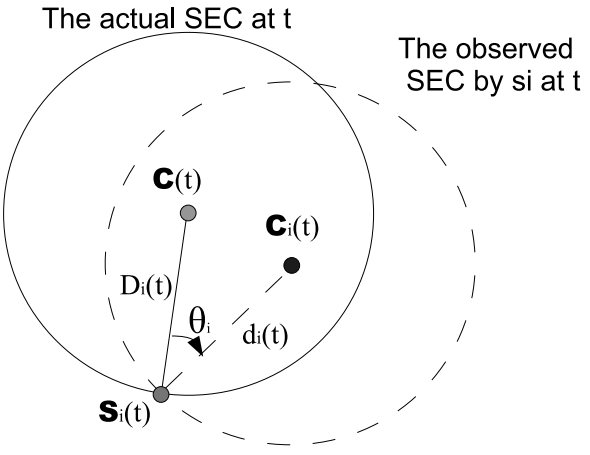


Fig. 8. Notations used in the proof

Lemma 1. *There is a constant* α $(0 \leq \alpha < 1)$ *that is independent of* t *and satisfies* $D'_i(\mathrm{t}) \leq \alpha D_i(\mathrm{t})$ *for any time* t *when a robot on the boundary of SEC is activated.*

Proof. Let θ_i be the angle error which s_i suffers at t. Since the distance traveled by s_i is $\frac{d_i(\mathrm{t})}{1+\epsilon_0}\cos\theta_0$, by applying the law of cosines to the triangle $\mathbf{C}(\mathrm{t})\mathbf{s_i}(\mathrm{t})\mathbf{s_i}(\mathrm{t}+1)$, we can obtain the following equation:

$$D'^2_i(\mathrm{t}) = D^2_i(\mathrm{t}) + \left(\frac{d_i(\mathrm{t})}{1+\epsilon_0}\right)^2 \cos^2\theta_0 - 2D_i(\mathrm{t})\frac{d_i(\mathrm{t})}{1+\epsilon_0}\cos\theta_0\cos\theta_i.$$

This equation can be transformed into the following inequality (the detail is shown in Appendix 1):

$$D'_i(\mathrm{t}) \leq D_i(\mathrm{t})\sqrt{1 + \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}\cos^2\theta_0}.$$

Since $0 \leq \epsilon_0 < 1$, the term of square root is less than one (see Appendix 2), and thus the lemma holds. □

Lemma 2. $R(\mathrm{t}+1) \leq \sqrt{R^2(\mathrm{t}) - \Delta^2(\mathrm{t})}$, *for all times* $t \geq 0$.

Proof. Lemma 1 implies that the destination of any movement is necessarily inside of the actual SEC. Thus, the radius of SEC is non-increasing. In addition, it is clear that the SEC at t (denoted by SEC_t) necessarily intersects that at $t+1$ (denoted by SEC_{t+1}). Thus, the following four cases are possible:

1. SEC_t and SEC_{t+1} have exactly one intersecting point and SEC_t does not contain SEC_{t+1}: All robots stay at their (unique) intersecting point, which implies $R(\mathrm{t}+1) = 0$.

2. SEC_t and SEC_{t+1} are identical: We can obtain $R(\mathrm{t}+1)=\sqrt{R^2(\mathrm{t})-\Delta^2(\mathrm{t})}=R(\mathrm{t})$ because $\Delta(\mathrm{t})=0$.

3. The boundaries of SEC_t and SEC_{t+1} have two intersecting points: Let $\mathbf{X_1}$ and $\mathbf{X_2}$ be the two intersecting points, and $\mathbf{Z}$ be the intersecting point of the lines $\mathbf{X_1X_2}$ and $\mathbf{C}(\mathrm{t})\mathbf{C}(\mathrm{t}+1)$. Letting L be the line passing through $\mathbf{C}(\mathrm{t}+1)$ and orthogonal to $\mathbf{C}(\mathrm{t})\mathbf{C}(\mathrm{t}+1)$, we define $\mathbf{Y_1}$ and $\mathbf{Y_2}$ as two intersecting points of L and the boundary of SEC_{t+1}. Notice that the segment $\mathbf{Y_1Y_2}$ is the diameter of SEC_{t+1}. We further divide this case into the following two sub-cases:
   (a) $dis(\mathbf{Z},\mathbf{C}(\mathrm{t}))<\Delta(\mathrm{t})$. (Fig.9)
   we show by contradiction that this case never occurs. Suppose $dis(\mathbf{Z},\mathbf{C}(\mathrm{t}))<\Delta(\mathrm{t})$ for contradiction. The circle C centered at $\mathbf{Z}$ and having diameter $dis(\mathbf{X_1},\mathbf{X_2})$ contains the intersecting area of SEC_t and SEC_{t+1}, and thus it encloses all robot locations at $t+1$. Since $\mathbf{Y_1Y_2}$ is the diameter of SEC_{t+1} and $dis(\mathbf{Z},\mathbf{C}(\mathrm{t}))<\Delta(\mathrm{t})$ holds, the length of $\mathbf{X_1X_2}$ is smaller than $R(\mathrm{t}+1)$. This contradicts to the fact that the SEC_{t+1} has the smallest diameter.
   (b) $dis(\mathbf{Z},\mathbf{C}(\mathrm{t}))\geq\Delta(\mathrm{t})$. (Fig.10)
   Let $\mathbf{Y_1'}$ and $\mathbf{Y_2'}$ be the two intersecting points of the line $\mathbf{Y_1Y_2}$ and the boundary of SEC_t. Then, $R(\mathrm{t}+1)=dis(\mathbf{Y_1},\mathbf{C}(\mathrm{t}))\leq dis(\mathbf{Y_1'},\mathbf{C}(\mathrm{t}+1))=\sqrt{R^2(\mathrm{t})-\Delta^2(\mathrm{t})}$ holds.
4. SEC_t contains SEC_{t+1}: This proof is the completely same as that for the case 3(b). (Fig.11)

From the above, the lemma holds for any cases. □

Lemma 3. *There exist two constants β and γ ($0<\beta,\gamma<1$, dependent on n but not on t) such that either of the followings holds for any time t when a robot s_i changes its position:*

- $R(\mathrm{t}+1)\leq\beta R(\mathrm{t})$.
- $\sum_i D_i(\mathrm{t}+1)\leq\gamma\sum_i D_i(\mathrm{t})$.

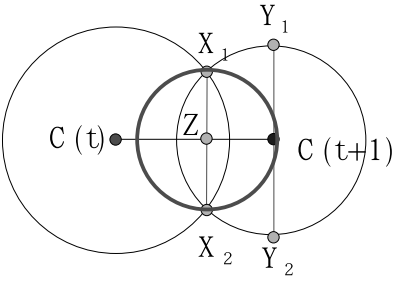


Fig. 9. $dis(Z,C(\mathrm{t}))<\Delta(\mathrm{t})$

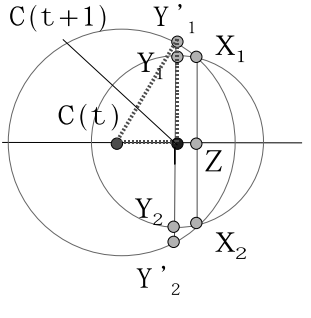


Fig. 10. $dis(Z,C(\mathrm{t}))\geq\Delta(\mathrm{t})$

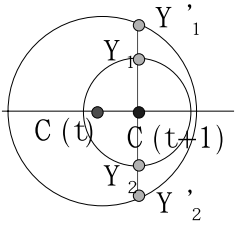


Fig. 11. The SEC at time $t+1$ exists in the SEC at time t

Proof. Let $S'(\mathrm{t})$ is the set of robots moving at time t. By the triangle inequality, $dis(\mathbf{P}, \mathbf{C}(\mathrm{t})) + \Delta(\mathrm{t}) \geq dis(\mathbf{P}, \mathbf{C}(\mathrm{t}+1))$ holds for any point $\mathbf{P}$. Combining this inequality with Lemma 1, we obtain $\alpha D_i(\mathrm{t}) + \Delta(\mathrm{t}) \geq D_i(\mathrm{t}+1)$. It follows the inequality below:

$$\begin{aligned}
&\sum_i D_i(\mathrm{t}+1) \\
&\leq \sum_{i \notin S'(\mathrm{t})} (D_i(\mathrm{t}) + \Delta(\mathrm{t})) + \sum_{i \in S'(\mathrm{t})} (\alpha D_i(\mathrm{t}) + \Delta(\mathrm{t})) \\
&= \sum_i D_i(\mathrm{t}) + n\Delta(\mathrm{t}) - (1-\alpha) \sum_{i \in S'(\mathrm{t})} D_i(\mathrm{t}).
\end{aligned}$$

Since only the robots on the boundary of SEC can move, $\sum_{i \in S'(\mathrm{t})} D_i(\mathrm{t}) = |S'(\mathrm{t})|R(\mathrm{t})$ holds for any t, and thus we have $\sum_i D_i(\mathrm{t}+1) \leq \sum_i D_i(\mathrm{t}) + n\Delta(\mathrm{t}) - (1-\alpha)|S'(\mathrm{t})|R(\mathrm{t})$. Then we consider the following two cases:

1. $n\Delta(\mathrm{t}) - (1-\alpha)|S'(\mathrm{t})|R(\mathrm{t}) > \frac{\alpha-1}{2n} \sum_i D_i(\mathrm{t})$.
   In this case, we can obtain inequality $\Delta(\mathrm{t}) > \frac{2(1-\alpha)|S'(\mathrm{t})|R(\mathrm{t}) + \frac{(\alpha-1)}{n} \sum_i D_i(\mathrm{t})}{2n}$. Moreover, $\sum_i D_i(\mathrm{t}) \leq nR(\mathrm{t})$ clearly holds. Thus, we have

$$\begin{aligned}
\Delta(\mathrm{t}) &> (1-\alpha)\frac{2|S'(\mathrm{t})|R(\mathrm{t}) - R(\mathrm{t})}{2n} \\
&= (1-\alpha)R(\mathrm{t})\frac{2|S'(\mathrm{t})| - 1}{2n}.
\end{aligned}$$

   From Lemma 2, we obtain

$$\begin{aligned}
R(\mathrm{t}+1) &< \sqrt{R^2(\mathrm{t}) - \left((1-\alpha)R(\mathrm{t})\frac{2|S'(\mathrm{t})| - 1}{2n}\right)^2} \\
&= R(\mathrm{t})\sqrt{1 - \left(\frac{(2|S'(\mathrm{t})| - 1)(1-\alpha)}{2n}\right)^2}.
\end{aligned}$$

   Since $1 \leq |S'(\mathrm{t})| \leq n$, $R(\mathrm{t}+1) < R(\mathrm{t})\sqrt{1 - \left(\frac{1-\alpha}{2n}\right)^2}$ holds. The term of the square root is smaller than one because $0 \leq \alpha < 1$. Thus, the lemma holds.

2. $n\Delta(\mathrm{t}) - (1-\alpha)|S'(\mathrm{t})|R(\mathrm{t}) \leq \frac{\alpha-1}{2n}\sum_i D_i(\mathrm{t})$.
   Then, we can obtain

$$\sum_i D_i(\mathrm{t}+1) \leq \sum_i D_i(\mathrm{t}) + \frac{\alpha-1}{2n}\sum_i D_i(\mathrm{t}) = \frac{2n-1+\alpha}{2n}\sum_i D_i(\mathrm{t}).$$

   This implies $\gamma = (2n-(1-\alpha))/2n$, and thus the lemma holds. □

Theorem 2. *In the uniform error model satisfying $\theta_0 < \pi/2$ and $0 \leq \epsilon_0 < 1$, algorithm Conv-SEC can converge all robots into a point.*

Proof. From Lemma 3, when a robot changes its position, either the radius of SEC or the sum of the distances between the center of SEC and all robots is decreased by a constant fraction (not depending on t). It implies that at least one of them converges to zero during infinite executions. Thus, the distance between any two robots converges to zero. □

5 Conclusion

In this paper, we newly introduced the notion of uniformity in observation error of autonomous mobile robots, and investigated the impact of uniformity to the solvability of the convergence problem. We showed that in the uniform error model with the maximum angle error $\epsilon_0 \geq \pi/2$, no algorithm can achieve the converge. In addition, assuming the semi-synchronous model and uniform observation error with $0 \leq \epsilon_0 < 1$ and $\theta_0 < \pi/2$, we proposed a convergence algorithm correctly working for any number of robots. This algorithm is optimal in the sense of its allowable error angle.

Acknowledgement

This work is supported by the Japan Society for the Promotion of Science: Grant-in-Aid for Young Scientists(B)(19700058), Grant-in-Aid for Scientific Research(C)(21500013), (C)(21500012), The Telecommunication Advancement Foundation, and Hori Information Science Promotion Foundation.

References

1. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distreibuted memoryless point convergence algorithm for mobile robots with lemited visibility. IEEE Transactions on Robotics and Automation 15(5), 818–828 (1999)
2. 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)
3. Cohen, R., Peleg, D.: Convergence properties of the gravitational algorithm in asynchronous robot systems. SIAM Journal on Computing 34(6), 1516–1528 (2005)

4. Cohen, R., Peleg, D.: Convergence of autonomous mobile robots with inaccurate sensors and movement. SIAM Journal on Computing 38, 276–302 (2008)
5. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theoretical Computer Science 337(1-3), 147–168 (2005)
6. Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: Gathering autonomous mobile robots with dynamic compasses: An optimal result. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 298–312. Springer, Heidelberg (2007)
7. Katayama, Y., Tomida, Y., Imazu, H., Inuzuka, N., Wada, K.: Dynamic compass models and gathering algorithms for autonomous mobile robots. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 274–288. Springer, Heidelberg (2007)
8. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theoretical Computer Science 384(2-3), 222–231 (2007)
9. Souissi, S., Defago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 333–349. Springer, Heidelberg (2006)
10. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal on Computing 28(4), 1347–1363 (1999)

Appendix 1

The omitted expression transformation in Lemma 1.

$$
\begin{aligned}
& D_i'^2(\mathrm{t}) \\
&= D_i^2(\mathrm{t}) + \left(\frac{d_i(\mathrm{t})}{1+\epsilon_0}\right)^2 \cos^2\theta_0 - 2D_i(\mathrm{t})\frac{d_i(\mathrm{t})}{1+\epsilon_0}\cos\theta_0\cos\theta_i \\
&\le D_i^2(\mathrm{t}) + \left(\frac{d_i(\mathrm{t})}{1+\epsilon_0}\right)^2 \cos^2\theta_0 - 2D_i(\mathrm{t})\frac{d_i(\mathrm{t})}{1+\epsilon_0}\cos^2\theta_0 \\
&= D_i^2(\mathrm{t}) + \left(\left(\frac{d_i(\mathrm{t})}{1+\epsilon_0}\right)^2 - 2D_i(\mathrm{t})\left(\frac{d_i(\mathrm{t})}{1+\epsilon_0}\right)\right)\cos^2\theta_0 \\
&= D_i^2(\mathrm{t}) + \left(\left(\frac{d_i(\mathrm{t})}{1+\epsilon_0}\right) - D_i(\mathrm{t})\right)^2 \cos^2\theta_0 - D_i^2(\mathrm{t})\cos^2\theta_0 \\
&\le D_i^2(\mathrm{t}) + \left(\frac{1-\epsilon_0}{1+\epsilon_0}D_i(\mathrm{t}) - D_i(\mathrm{t})\right)^2 \cos^2\theta_0 - D_i^2(\mathrm{t})\cos^2\theta_0 \\
&= D_i^2(\mathrm{t})\left(1 + \left(\frac{1-\epsilon_0}{1+\epsilon_0} - 1\right)^2 \cos^2\theta_0 - \cos^2\theta_0\right) \\
&= D_i^2(\mathrm{t})\left(1 + \frac{3\epsilon_0^2 - 2\epsilon_0 - 1}{(1+\epsilon_0)^2}\cos^2\theta_0\right) \\
&= D_i^2(\mathrm{t})\left(1 + \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}\cos^2\theta_0\right). \qquad \square
\end{aligned}
$$

Appendix 2

The omitted proof for $0 \leq 1 + \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2} \cos^2\theta_0 < 1$.
The differential of the term $\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}$ with respect to ϵ_0 is as follows:

$$\frac{(6\epsilon_0 - 2)(1+\epsilon_0)^2 - \left(3\epsilon_0^2 - 2\epsilon_0 - 1\right) 2(\epsilon_0 + 1)}{(1+\epsilon_0)^4} = \frac{8\epsilon_0}{(1+\epsilon_0)^3}$$

The term $\frac{8\epsilon_0}{(1+\epsilon_0)^3}$ is nonnegative for $0 \leq \epsilon_0 < 1$. It shows that $\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}$ is nondecreasing. Hence, its minimum value is -1 for $\epsilon_0 = 0$ and the minimum is zero for $\epsilon_0 = 1$. Since $0 < \cos^2\theta_0 \leq 1$ holds for $\theta_0 < \pi/2$, we consequently have $0 \leq 1 + \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2} \cos^2\theta_0 < 1$. □

An Optimal Bit Complexity Randomized Distributed MIS Algorithm (Extended Abstract)*

Métivier Yves, John Michael Robson, Saheb-Djahromi Nasser, and Akka Zemmari

LaBRI - Université de Bordeaux, CNRS
351 cours de la Libération. 33405 Talence, France
{metivier,robson,saheb,zemmari}@labri.fr|

Abstract. We present a randomized distributed maximal independent set (MIS) algorithm for arbitrary graphs of size n that halts in time $O(\log n)$ with probability $1 - o(n^{-1})$, each message containing 1 bit: thus its bit complexity per channel is $O(\log n)$ (the bit complexity is the number of bits we need to solve a distributed task, it measures the communication complexity). We assume that the graph is anonymous: unique identities are not available to distinguish the processes; we only assume that each vertex distinguishes between its neighbours by locally known channel names. Furthermore we do not assume that the size (or an upper bound on the size) of the graph is known. This algorithm is optimal (modulo a multiplicative constant) for the bit complexity and improves the best previous randomized distributed MIS algorithms (deduced from the randomized PRAM algorithm due to Luby [Lub86]) for general graphs which is $O(\log^2 n)$ per channel (it halts in time $O(\log n)$ and the size of each message is $\log n$). This result is based on a powerful and general technique for converting unrealistic exchanges of messages containing real numbers drawn at random on each vertex of a network into exchanges of bits. Then we consider a natural question: what is the impact of a vertex inclusion in the MIS on distant vertices? We prove that this impact vanishes rapidly as the distance grows for bounded-degree vertices. We provide a counter-example that shows this result does not hold in general. We prove also that these results remain valid for Luby's algorithm presented by Lynch [Lyn96] and by Wattenhofer [Wat07]. This question remains open for the variant given by Peleg [Pel00].

1 Introduction

1.1 The Problem

Let $G = (V, E)$ be a simple connected undirected graph. An independent set of G is a subset I of V such that no two members of I are adjacent. An independent set I is maximal if any vertex of G is in I or adjacent to a vertex of I.

* This work was supported by grant No ANR-06-SETI-015-03 awarded by Agence Nationale de la Recherche.

S. Kutten and J. Žerovnik (Eds.): SIROCCO 2009, LNCS 5869, pp. 323–337, 2010.

In this paper we discuss how greedy selection for the computation of a maximal independent set (MIS) in a network of processors can be accomplished by exchange of messages between adjacent processors. The distributed complexity (time and bit) of computing an MIS is of fundamental interest for the study and the analysis of distributed computing. The computation of an MIS is a building block for many distributed algorithms: topology control, routing, coloring. Thus an MIS provides an initial clustering which can be used as an initial structure. Furthermore an MIS induces a set of processors which can operate in parallel. A presentation and results concerning the MIS problem can be found in [Lyn96] (Chapter 4, p. 71-76) and in [Pel00] (Chapter 8).

1.2 The Model

The Network. We consider the standard message passing model for distributed computing. The communication model consists of a point-to-point communication network described by a simple connected undirected graph $G = (V, E)$ where the vertices V represent network processors and the edges represent bidirectional communication channels. Processes communicate by message passing: a process sends a message to another by depositing the message in the corresponding channel. We assume the system synchronous and synchronous wake-up of processors: processors have access to a global clock and all processors start the algorithm at the same time.

Time Complexity. A round (cycle) of each processor is composed of the following three steps: 1. Send messages to (some of) the neighbours, 2. Receive messages from (some of) the neighbours, 3. Perform some local computation. As usual (see for example Peleg [Pel00]) the time complexity is the maximum possible number of rounds needed until every node has completed its computation.

Bit Complexity. As is explained by Santoro in [San07] (Chapter 6) (see also [Gho06], Chapter 3) the cost of a synchronous distributed algorithm is both time and bits. By definition, the bit complexity of a distributed algorithm (per channel) is the total number of bits exchanged (per channel) during its execution. Thus it is considered as a finer measure of communication complexity and it has been studied for breaking and achieving symmetry or for coloring in [BMW94, KOSS06, DMR08]. Dinitz et al. explain in [DMR08] that it may be viewed as a natural extension of communication complexity (introduced by Yao [Yao79]) to the analysis of tasks in a distributed setting. An introduction to this area can be found in Kushilevitz and Nisan [KN99].

Network and Processes Knowledge. The network is anonymous: unique identities are not available to distinguish the processes. We do not assume any global knowledge of the network, not even its size or an upper bound on its size. The processors do not require any position or distance information. Each processor knows from which channel it receives a message. An important fact (see [Pel00] P. 93) due to the initial symmetry is: *there is no deterministic distributed*

algorithm for arbitrary anonymous graphs for computing an MIS assuming all vertices wake up simultaneously.

1.3 Our Contribution

As for a number of randomized algorithms for MIS, our algorithms work in the following way. They operate in synchronous rounds grouped into phases. At the end of each phase, some vertices join the MIS and some others know they will never be in the MIS: these two sets of vertices erase themselves from the graph.

One of the main contributions of this paper is the development of a powerful and general technique for converting unrealistic exchanges of messages containing real numbers drawn at random on each vertex inqto efficient exchanges of bits, drawn at random on each vertex, and we apply this technique to the computation of an MIS.

First we consider in Section 2 the model of exchange of messages containing real numbers and we deduce a very simple randomized algorithm (Algorithm $\mathcal{A}$) for the computation of an MIS. We derive logarithmic bounds on the number of exchanges required; one such bound on the average and another which holds with probability $1 - o(n^{-1})$; we deduce that Algorithm $\mathcal{A}$ computes an MIS for arbitrary graphs of size n in time $O(\log n)$ with probability $1 - o(n^{-1})$.

Then we discuss in Section 3 how, in the model of exchange of single bit messages, the real number exchanges can be simulated in finite time (Algorithm $\mathcal{B}$) and we show logarithmic bounds on the number of bits used to complete all the real exchanges and finally Algorithm $\mathcal{B}$ computes an MIS for arbitrary graphs of size n in time $O(\log^2 n)$ with probability $1 - o(n^{-1})$.

In Section 4 we show how the simulated real number exchanges can be overlapped (Algorithm $\mathcal{C}$), in this way we prove Theorem 3:

There exists a randomized distributed MIS algorithm for arbitrary graphs of size n that halts in time $O(\log n)$ with probability $1 - o(n^{-1})$, each message containing 1 bit.

We conclude that the bit complexity per channel of algorithm $\mathcal{C}$ is $O(\log n)$.

Algorithm $\mathcal{C}$ is optimal for the bit complexity as a direct consequence of two results. First, Kothapalli et al. show in [KOSS06] that if only one bit can be sent along each edge in a round, then every distributed vertex colouring algorithm (in which every node has the same initial state and initially only knows its own edges) needs at least $\Omega(\log n)$ rounds with high probability[1] (w.h.p. for short) to colour the cycle of size n with any finite number of colours. Second, Wattenhofer in [Wat07] (p.36) shows that a colouring algorithm can be obtained from an MIS algorithm. The colouring algorithm ([Wat07] Algorithm 18) has the same (time and bit) complexity as the MIS algorithm, up to a multiplicative constant, for any family of graphs of bounded degree and in particular for the family of rings.

A fundamental question about distributed computation is: *What can (or cannot) be computed locally?* (see [NS95, KMW04]). In the context of randomized

[1] With high probability means with probability $1 - o(n^{-1})$.

distributed algorithms, we may consider a linked natural question: what is the impact of a vertex inclusion in the MIS on distant vertices?

We prove in Section 5 that this impact vanishes rapidly as the distance grows for bounded-degree vertices. We provide a counter-example that shows this result does not hold in general. We prove also that these results remain valid for Luby's algorithms presented by Lynch [Lyn96] and by Wattenhofer [Wat07] and described in the next subsection. This question remains open for the variant given by Peleg [Pel00].

Remark 1. Proofs are omitted due to lack of space.

1.4 Related Works: Comparisons and Comments

The computation of an MIS has been the object of extensive research on parallel and distributed complexity [ABI86, Lub86, AGLP89, Lin92]; Karp and Widgerson [KW84] have proved that the MIS problem is in NC. Some links with distributed graph coloring and some recent results on this problem can be found in [KW06]. The complexity of some special classes of graphs such as growth-bounded graphs is studied in [KMNW05]. Results have been obtained also for radio networks [MW05].

A major contribution is due to Luby [Lub86]: he presented a randomized PRAM algorithm which requires a linear number of processors and runs in $O(\log^2 n)$. He assumes that the number of vertices is known. The main idea is to obtain for each vertex a *local total order* or a local election which breaks the local symmetry and then each vertex can decide locally whether it joins the MIS or not.

In the context of distributed computation, a first application of this idea corresponds to algorithm $\mathcal{A}$ of this paper and to the presentation of Luby's algorithm (called LubyMIS) given by Lynch [Lyn96] (Chapter 4, p. 71-76). In this presentation, the knowledge of the size n of the network is used to enable each vertex to make a random choice of an integer from $\{0, \ldots, n^4\}$ using the uniform distribution. The analysis of this presentation may be summarised by (Theorem 4.9, p. 76): *With probability one, LubyMIS eventually terminates. Moreover, the expected number of rounds until termination is* $O(\log n)$. Each message contains $O(\log n)$ bits thus the bit complexity per channel is $O(\log^2 n)$.

A variant of Luby's algorithm is presented in [Pel00] (Chapter 8) which allows a simpler analysis. The local election is done in the following way: at each phase, each vertex v computes the maximal vertex degree of its 2-neighbourhood $D(v)$ (vertices at distance 1 or 2) and then draws uniformly at random a bit with probability depending on $D(v)$. If v draws 1 and every neighbour draws 0 then v joins the MIS. Peleg shows that this algorithm halts in time $O(\log^2 n)$ with probability $1-o(n^{-1})$. It needs messages containing $\log n$ bits thus the bit complexity per channel is $O(\log^3 n)$.

Wattenhofer [Wat07] presents and analyses another distributed implementation of Luby's algorithm. Each vertex needs at each phase the knowledge of the degrees of its neighbours, and marks itself with probability $1/(2d(v))$, where $d(v)$

is the current degree of v. If no higher degree neighbour of v is also marked then v joins the MIS. If a higher degree neighbour of v is marked v unmarks itself. If the neighbours have the same degree ties are broken arbitrarily. This algorithm terminates in expected $O(\log n)$ time and the size of messages is $\log n$: its bit complexity per channel is $O(\log^2 n)$.

Alon et al. [ABI86] presents a parallel randomized algorithm to find an MIS, its expected time on a PRAM is $O(\log n)$. As for the previous one, each vertex needs at each phase the degrees of its neighbours: its bit complexity is $O(\log^2 n)$.

As is explained in [San07] (p. 18), the knowledge of vertices in distributed computing is fundamental. For example, there exists a deterministic election algorithm for an anonymous network minimal for the covering relation if the size or a bound on the size is known and no such algorithm exists if the size (or a bound) is not known. In the same way, it is shown that it is possible to break the symmetry in anonymous networks but that it is not possible to detect termination unless the network size is known. About the computation of the network size one can cite the following impossibility result for anonymous networks. There exists no process-terminating algorithm for computing the size that is correct with probability $r > 0$. ([Tel00], Chapter 9).

The table below summarises the comparison between the various MIS algorithms and Algorithm $\mathcal{C}$ of this paper.

	Knowledge	Time	Message size (number of bits)	Bit complexity (per channel)
Luby (Lynch)	Size of the graph	$O(\log n)$	$\log n$	$O(\log^2 n)$
Luby (Peleg)	Maximum degree in 2-Neighbourhood	$O(\log^2 n)$	$\log n$	$O(\log^3 n)$
Luby (Wattenhofer)	Maximum of neighbours degrees	$O(\log n)$	$\log n$	$O(\log^2 n)$
Alon, Babai and Itai	Maximum of neighbours degrees	$O(\log n)$	$\log n$	$O(\log^2 n)$
Algorithm $\mathcal{C}$	no knowledge	$O(\log n)$	1	$O(\log n)$

Notation. In this paper, as usual, $Pr(e)$ denotes the probability of the event e and $E(X)$ the expected value of the random variable (r.v.) X.

2 Exchange of Real Numbers

The first distributed algorithm, denoted $\mathcal{A}$, is very simple. It is composed of phases. At each phase each processor u still in the graph generates a random variable $x(u)$ and a processor is included in the independent set if its x is a local minimum, i.e., $x(u) < x(v)$ for each neighbour v of u. (Algorithm 1 describes a phase of Algorithm $\mathcal{A}$) It is convenient to take the random variables to be uniformly distributed on $[0, 1)$.

When all local minima have been included in the independent set, each surviving processor (not included in the MIS and not definitely excluded from the MIS) generates a new random variable and again local minima are included in the independent set. The algorithm halts when there is no surviving processor.

The outcome of an MIS computation is defined on each vertex u by a special variable $\eta(u)$. A vertex u joining the MIS sets $\eta(u)$ to 1 and a vertex u not joining the MIS sets $\eta(u)$ to 0; initially $\eta(u) = -1$.

For each vertex v of the graph: $Not\text{-}In\text{-}MIS\text{-}set(v)$ and $In\text{-}MIS\text{-}set(v)$ are sets of vertices, initially $Not\text{-}In\text{-}MIS\text{-}set(v) = \emptyset$, $In\text{-}MIS\text{-}set(v) = \emptyset$.

Draw uniformly at random a real $x(u)$;
Send $x(u)$ to all neighbours w;
Receive $x(w)$ from all neighbours w;
if $(x(u) < x(w)$ for each neigbour of $u)$ **then**
  Set $\eta(u) = 1$;
  Send $In\text{-}MIS$ to each neighbour;
end if
Receive a message $mess(w)$ from all neighbours w;
Put w in $In\text{-}MIS\text{-}set(u)$ for each neighbour w such that $(mess(w) = In\text{-}MIS)$;
if $mess(w) = In\text{-}MIS$ for at least one neighbour w **then**
  Set $\eta(u) = 0$;
  Send $Not\text{-}In\text{-}MIS$ to all neighbours;
end if
Receive a message $mess(w)$ from all neighbours w;
Put w in $Not\text{-}In\text{-}MIS\text{-}set(u)$ for each neighbour w such that $(mess(w) = Not\text{-}In\text{-}MIS)$;
Erase from neighbours of u in the graph each vertex w in $In\text{-}MIS\text{-}set(u)$ or in $Not\text{-}In\text{-}MIS\text{-}set(u)$;

Algorithm 1. A phase of Algorithm $\mathcal{A}$

We have the following lemma:

Lemma 1. *In any phase, the expected number of edges removed from the remaining graph G is at least half the number of edges in G.*

Proof. We say that a vertex u *preemptively removes* a neighbour v if $x(u)$ is less than $x(v)$ and $x(w)$ for all other neighbours w of u and v. If this is the case, then u will be included in the independent set and so v and all edges (v, w) incident on v will be removed from the graph. We say that the edges (v, w) are *preemptively removed*. If the degrees are $d(u)$ and $d(v)$, the probability that u preemptively removes v is at least $1/(d(u) + d(v))$. The average number of edges preemptively removed is thus at least $(\sum_{(u,v)\in E}(\frac{d(v)}{d(u)+d(v)} + \frac{d(u)}{d(u)+d(v)}))/2$ since $d(v)$ are removed if u removes v and $d(u)$ are removed if v removes u and an edge (v, w) can only be preemptively removed twice, once by the removal of v and once by that of w. The sum is $(\sum_{(u,v)\in E} 1)/2$, that is half the number of edges.

Remark 2. We introduce the notion of "to be preemptively removed" for the proof of this lemma and thanks to this analysis, a vertex does not need in the sequel of this paper to know the maximal vertex degree of its 2-neighbourhood.

We then obtain :

Corollary 1. *There are constants k_1 and K_1 such that for any graph $G = (V, E)$ of n vertices the number of phases to remove all edges from G is less than $k_1 \log n$ on average, and less than $K_1 \log n$ with probability $1 - o(n^{-1})$.*

These results are summarised by:

Theorem 1. *Algorithm $\mathcal{A}$ computes an MIS for arbitrary graphs of size n in time $O(\log n)$ with probability $1 - o(n^{-1})$.*

3 Exchange of Bits in Phases

The main idea. In this section we present and analyse an algorithm which simulates exchanges of real numbers by exchanges of bits which define real numbers, most significant first.

3.1 Description of the Algorithm

We consider a more realistic algorithm, denoted $\mathcal{B}$, in which processors exchange messages of finite size; in fact we consider only messages of a fixed finite set of types: one bit 0 or 1 of Data, *In-MIS*,*Not-In-MIS*, and *Ineligible* (when a vertex v sends this message it means that until the end of the current phase v cannot be in the MIS).

At the start of a phase, a processor knows which neighbours are still in the graph and initialises a set of *active* neighbours to all of these. The status of a vertex may be *Eligible:* the vertex may be included in the independent set during this phase, or *Ineligible:* the vertex cannot be included in the independent set during this phase. All processors still in the graph are initially *Eligible.*

One phase of exchange of real numbers is replaced by a phase composed of a sequence of rounds. Each round is composed of send, receive and internal actions. A message is one of these types. The *Data* messages send the bits of a real number, most significant first. The other messages permit a processor to stop sending *Data* when the real numbers are known accurately enough for further bits to be irrelevant to whether any processor has a local minimum (the exact simulation of the order induced by real numbers implies that during the course of a phase, two neighbours exchange bits as long as they have not determined whether one of them is or is not a local minimum even if other information implies that one of them is no longer eligible).

In each round each processor u generates one random bit and sends it to each active neighbour v and then does the following:

- If its bit was 0 and it received 1 from each active neighbour, it is a local minimum. It sets $\eta(u)$ to 1 and it sends *In-MIS* to all neighbours and takes no further part in this or later phases.
- If its bit was different from that received from v, it removes v from its list of active neighbours for this phase (the symmetry is broken; now the order relation between the two vertices is known thus they do not need to exchange more bits).
- If its bit was 1 and it received 0 from a neighbour, it is not a local minimum. Its status, for this phase, becomes *Ineligible* and it sends *Ineligible* to every active neighbour and removes from its active list all ineligible neighbours.
- If it receives *In-MIS* from any neighbour, it sets $\eta(u)$ to 0 and it sends *Not-In-MIS* to all other neighbours. It takes no part in subsequent phases but continues to generate and send bits as long as it has active neighbours.
- If it receives *Not-In-MIS* from a neighbour v, it notes that v is not eligible and will be removed from the graph after this phase. If it is itself ineligible, it removes v from its active list.
- If it receives *Ineligible* from a neighbour v, it notes that v is not eligible in this phase and, if it is itself ineligible, removes v from its active list.
- If it is ineligible and has no eligible active neighbours it is the end of the current phase. It takes no part in this phase and is ready to start another phase if $\eta(u) = -1$.

Initially, for each vertex v of the graph: $\eta(v) = -1$; and $active\text{-}set(v)$, $Not\text{-}In\text{-}MIS\text{-}set(v)$ and $In\text{-}MIS\text{-}set(v)$ are sets of vertices. At the beginning of each phase, for each vertex v such that $\eta(v) = -1 : status(v) = Eligible, active\text{-}set(v)$ contains the set of neighbours w of v satisfying $\eta(w) = -1$, and $Not\text{-}In\text{-}MIS\text{-}set(v) = \emptyset$, $In\text{-}MIS\text{-}set(v) = \emptyset$ and $Ineligible\text{-}set(v) = \emptyset$.

It follows from this:

Remark 3. After an exchange of bits and all consequent other messages, u and v consider each other as active neighbours if and only if they have generated exactly the same sequence of bits so far in this phase and one of them is still eligible.

This is precisely the condition that they still need to exchange more bits to decide whether one of them is a local minimum.

3.2 Analysis of the Algorithm

Lemma 2. *In any round of exchange of bits, any processor u has probability at least $1/4$ of entering one of the following states (if it has not already done so): $In\text{-}MIS$, $Not\text{-}In\text{-}MIS$ and $End\text{-}of\text{-}phase$.*

Then:

Corollary 2. *There exist constants k_2 and K_2 such that the maximum number of Data bits generated by any processor u in all phases is less than $k_2 \log n$ on average and less than $K_2 \log n$ with probability $1 - o(n^{-2})$.*

while ($\eta(v) \neq 1$ and $\eta(v) \neq 0$ and *active-set*(v) is not empty) **do**
  Draw uniformly at random a bit $b(v)$;
  Send $b(v)$ to all active neighbours w;
  Receive $b(w)$ from all active neighbours w;
  if $b(v) = 0$ **then**
    if all active neighbours w have drawn $b(w) = 1$ **then**
      Set $\eta(v) = 1$;
      Send $In\text{-}MIS$ to each neighbour;
    end if
  else
    if there is at least one active neighbour w which has drawn $b(w) = 0$ **then**
      Set $status(v) = Ineligible$;
      Send $Ineligible$ to each active neighbour;
    end if
  end if
  Receive a message $mess(w)$ from all active neighbours w;
  Put w in $Ineligible\text{-}set(v)$ for each neighbour w such that $(mess(w) = Ineligible)$;
  Put w in $In\text{-}MIS\text{-}set(v)$ for each neighbour w such that $(mess(w) = In\text{-}MIS)$;
  if $mess(w) = In\text{-}MIS$ for at least one neighbour w **then**
    Set $\eta(v) = 0$;
    Send $Not\text{-}In\text{-}MIS$ to all neighbours;
  end if
  Receive a message $mess(w)$ from all active neighbours w;
  Put w in $Not\text{-}In\text{-}MIS\text{-}set(v)$ for each neighbour w such that $(mess(w) = Not\text{-}In\text{-}MIS)$;
  if $status(v) = Ineligible$ **then**
    Remove w from *active-set*(v) for each neighbour w in $Not\text{-}In\text{-}MIS\text{-}set(v)$ or in $In\text{-}MIS\text{-}set(v)$ or in $Ineligible\text{-}set(v)$ and for each neighbour w such that $b(v) \neq b(w)$;
  end if
end while
Erase from neighbours of v in the graph each vertex w in $In\text{-}MIS\text{-}set(v)$ or in $Not\text{-}In\text{-}MIS\text{-}set(v)$;

Algorithm 2. A phase of Algorithm $\mathcal{B}$

Yielding:

Corollary 3. *There exists k_3 such that the maximum number of Data bits generated by any processor is less than $k_3 \log n$ on average and with probability $1 - o(n^{-1})$.*

These results are summarised by:

Theorem 2. *Algorithm $\mathcal{B}$ computes an MIS for arbitrary graphs of size n in time $O(log^2 n)$ with probability $1 - o(n^{-1})$.*

4 Exchange of Bits with Desynchronised Phases between Adjacent Edges

We present a method of simulating Algorithm $\mathcal{B}$ by means of 1-bit messages sent along edges between neighbouring vertices, obtaining a new algorithm: Algorithm $\mathcal{C}$. The maximum number of messages sent and received by any vertex will be $O(\log n)$ on average and with high $(1 - o(n^{-1}))$ probability.

The main idea. Algorithm $\mathcal{B}$ simulates exchange of real numbers by exchange of bits. The exchange of bits is centralised on the vertex: the vertex exchanges bits (corresponding to a real number) with neighbours until the symmetry is broken: at each round a vertex sends the same bit to all its neighbours. In this section, the main idea is, for each vertex u, a desynchronisation between edges incident on u: the vertex u exchanges bits with a given neighbour v until the symmetry is broken between u and v. If, in a round, u breaks the symmetry with v_1 and does not break the symmetry with v_2 then u considers that a phase with v_1 is completed and starts, in anticipation, a new phase with v_1 and it continues the previous phase with v_2. When a bit is drawn by anticipation, it is memorized to be used later by another edge when it accomplishes the same round in the same phase. Thus, in the same round, the vertex u may send the bit b_1 to the vertex v_1 corresponding to the phase t_1 and the bit b_2 to v_2 corresponding to the phase t_2 with $t_1 \neq t_2$.

Remark 4. The fundamental fact is that each round has probability 1/2 of breaking the symmetry over an edge.

General description of the algorithm on each vertex. Each vertex runs two processes in interleaved fashion, alternating one round of process **calc_win** and one round of **calc_mis**. The process **calc_win** computes for each pair of neighbouring vertices and for each phase, which of the two has the smaller value in the phase. The process **calc_mis** uses this information to find the MIS computed by the algorithm and eventually to halt the first process.

From time to time a vertex running **calc_mis** will decide that it is to be removed from the graph and signal this fact to its neighbours. Any reference to the neighbours of a vertex is to be understood to mean those neighbours from whom no such signal has yet been received.

With the same notation defined previously, initially, for each vertex u of the graph: $\eta(u) = -1$; and *active-set*(v), $Not\text{-}In\text{-}MIS\text{-}set(v)$ and $In\text{-}MIS\text{-}set(v)$ are sets of vertices. The variable *active-set*(u) contains the set of neighbours v of u satisfying $\eta(v) = -1$.

while ($\eta(u) \neq 1$ and $\eta(u) \neq 0$ and *active-set*(u) is not empty) **do**
  1 round of **calc_win**;
  1 round of **calc_mis**
4: **end while**

Algorithm 3. Algorithm $\mathcal{C}$

4.1 Variables of Algorithm $\mathcal{C}$

For each process u, the following variables are available:

- for each neighbour v of u, $phase_u(v)$ is a non negative integer, it is the number of the current phase between u and v; initially $phase_u(v)$ is equal to 1; by symmetry we have: $phase_u(v) = phase_v(u)$;
- for each neighbour v of u, $bit_u(v)$ is a non negative integer which denotes the number of the bit which will be sent by u to v in the current phase; initially, $bit_u(v)$ is equal to 1;
- X_u is a two dimensional array (notionally infinite), for each neighbour v of u and for an integer j $X_u[phase_u(v), j]$ is a bit;
- for each neighbour v of u and for each number t of a phase $win_u(v)(t)$ is a boolean which will be true if in phase t the symmetry is broken for the edge between u and v and u has the smaller value.

Let u be a vertex and let v be a neighbour of u; let t be the number of a phase; we denote by $x_u(v,t)$ the word defined by the bits sent by u to v since the beginning of the phase t. The length of the phase t is denoted by $\ell(t)$, it is equal to $Max\{|x(v,t)| \mid v \text{ is a neighbour of } u\}$, where $|x(v,t)|$ is the length of the word $x(v,t)$, initially, $\ell(t) = 0$.

Let u be a vertex, the phase t is active if there exists an active neighbour v of u such that $t = phase_u(v)$ and $x_u(v,t) = x_v(u,t)$.

4.2 Computing the *win* Bits: Process calc_win

At the beginning of a round of **calc_win**, the process u draws uniformly at random a new bit $b(t)$ for each active phase t on u, and puts it in X, i.e., $X[t, l(t)+1] := b(t)$. For each neighbour v, u will find $b(v) = X[phase_u(v), bit_u(v)]$, send $b(v)$ to v and receive the bit $b(u)$ from v. If $b(v) = b(u)$ then it is necessary to look at succeeding bits to distinguish $x_u(v, phase_u(v))$ and $x_v(u, phase_v(u))$ of phase $phase_u(v) = phase_v(u)$ thus u does $bit_u(v) := bit_u(v) + 1$. Else the result is recorded and the next phase can be considered; thus u does:
(1) $win_u(v)(phase_u(v)) := (b(v) = 0)$; (2) $phase_u(v) := phase_u(v) + 1$; and (3) $bit_u(v) := 1$.

4.3 Computing the MIS: Process calc_mis

For a vertex u, the computations for a phase t in which u is active take place in three stages. Initially u knows which neighbours v are active in the phase and it waits until it knows all its $win_u(v)(t)$ variables for them. Then it knows whether it is included in the MIS in the phase and sends an appropriate 1-bit *in* message to each v. In the second stage it waits until it has received an *in* message from each v. Then it knows whether it is excluded from the graph in the phase and sends an appropriate 1-bit *out* message to each v. In the final stage it waits until it has received an *out* message from each v. Then it knows which neighbours are active at the start of phase $t+1$. It updates $\eta(u)$ and its set of active neighbours and it is ready to start phase $t+1$ if it is still active.

4.4 Analysis of Algorithm $\mathcal{C}$

Remark 5. We know that on average and with high probability, T the number of phases is at most $k_1 \log n$. We conclude that after a number $O(\log n)$ of rounds all values $win_u(v)(t)$ of all vertices with $t \leq T$ have been computed since each round has probability $1/2$ of success on each edge (u, v). (Taking $K \log n$ rounds with K sufficiently large will give probability $o(n^{-3})$ of not having reached T for each edge (u, v) so that the probability that it happens for some edge is $o(n^{-1})$).

Now we give the main result of this section:

Theorem 3. *The randomized distributed MIS Algorithm $\mathcal{C}$ for arbitrary graphs of size n halts in time $O(\log n)$ with probability $1 - o(n^{-1})$, each message containing 1 bit.*

Finally:

Corollary 4. *The bit complexity per channel of Algorithm $\mathcal{C}$ is $O(\log n)$.*

5 Asymptotic Independence of Choices in MIS for Distant Vertices

5.1 Algorithm $\mathcal{C}$

Clearly the choice of a vertex inhibits that of the neighbouring ones and favours those at distance 2. A natural question would be: what is the impact of a vertex inclusion in the MIS on distant vertices? It is tempting to conjecture that the correlation vanishes rapidly as the distance grows. In the sequel we state this assertion for bounded-degree vertices. A counter-example is provided in a full version of the paper and shows that it *does not hold in general*.

Proposition 1. *Let u and v be two vertices at distance l in G. We suppose that they have finite fixed degrees. Let $Pr(v|u)$ denote the probability that v is chosen conditioned by u having been chosen and $Pr(v)$ the probability of the same event without conditioning. Then, as $l \to \infty$, $Pr(v|u) = Pr(v) + O(\delta^l)$, for some δ, with $|\delta| < 1$.*

Remark 6. It is easy to see that Proposition 1 holds under the weaker assumption that the degrees remain negligible with respect to the distance. However, the following example shows that the asymptotic independence of distant vertices *does not hold in general*.

Example 1. Consider the following graph G with two vertices u and v at distance $l = 4l' + 1$: the vertices are $\{u_{i,j}, v_{i,j} | i = 0, \cdots, 2l',\ j = 1, \cdots, l^{3i}\}$ (u is $u_{0,1}$ and v is $v_{0,1}$); the edges are $(u_{i,j}, u_{i,k})$, $(v_{i,j}, v_{i,k})$, $(u_{i,j}, u_{i+1,k})$, $(v_{i,j}, v_{i+1,k})$ and $(u_{2l',j}, v_{2l',k})$ for every i, j, k for which these vertices exist.

We call the *normal* history of the algorithm on G that in which:

- in phase 1, only one vertex is chosen, namely a $u_{2l',j}$ or $v_{2l',j}$; (we consider the case that it is a $u_{2l',j}$). This eliminates from the graph all vertices $u_{2l',k}$, $v_{2l',k}$ and $u_{2l'-1,k}$ (and no others).
- in phase i, $(1 < i < l')$, two vertices are chosen, one $u_{2(l'-i),j}$ and one $v_{2(l'-i)+1,k}$. This eliminates from the graph all vertices $u_{2(l'-i),m}$, $u_{2(l'-i)-1,m}$, $v_{2(l'-i)+1,m}$ and $v_{2(l'-i),m}$ (and no others).
- in phase l', u and one vertex $v_{1,j}$ are chosen eliminating all other vertices.

In the first phase, it is impossible that both a $u_{2l',j}$ and a $v_{2l',k}$ are chosen because they are all neighbours. The possibility of any vertex $u_{i,j}$ or $v_{i,j}$ being chosen in any phase other than according to the normal history is less than $l^{-3(i+1)}$ provided the normal history has been followed in previous phases because all its neighbours in level $i+1$ are still present. Summing over all i and j and all phases we find that the probability of any behaviour other than the normal history is $O(l^{-1})$. If the normal history is followed, either u or v but not both is chosen and by symmetry, each has the same probability. Thus $Pr(v) = Pr(u) = 1/2+O(l^{-1})$ but $Pr(u \textit{ and } v) = O(l^{-1})$.

The authors do not know at present any weaker condition under which the asymptotic independence of choices holds for distant vertices.

Remark 7. It is also possible to extend Proposition 1 for sets of independent vertices. Let U_1 and U_2 be two non empty sets of paiwise independent vertices of finite degrees. Let l denote the smallest distance between a member of U_1 and a member of U_2. Let us denote by $Pr(U_1)$ the probability that *all* members of U_1 are chosen in the MIS and by $Pr(U_1|U_2)$ the probability of the same event conditioned the event that *all* members of U_2 are. Then, as $l \to \infty$: $Pr(U_1|U_2) = Pr(U_1) + O(\delta^l)$, for some δ, with $|\delta| < 1$.

5.2 Luby's Algorithms

The above result will remain valid for any algorithm which obeys two conditions:

1. The behaviour of the agorithm is *local*: in each phase the actions at a vertex depend only on the decisions within a ball of fixed radius and affect only vertices within this same ball.
2. Bounding of the probability of removal: a vertex of initial degree d has probability at least ϵ/d of being removed from the graph in each phase until its removal.

The second condition is called a lower bounding property. In this subsection, we prove that Luby's algorithms presented by Lynch and by Wattenhofer satisfy a lower bounding property and thus Proposition 1 remains valid.

Lemma 3. *The Luby (Lynch) algorithm satisfies a lower bounding property for* $\epsilon = 1/3$.

Lemma 4. *The Luby (Wattenhofer) algorithm satisfies a lower bounding property for* $\epsilon = 1/4$.

Acknowledgements. We are grateful to David Peleg and Roger Wattenhofer for helpful and enlightening e-discussions.

References

[ABI86] Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set. Journal of Algorithms 7(4), 567–583 (1986)

[AGLP89] Awerbuch, B., Goldberg, A.V., Luby, M., Plotkin, S.A.: Network decomposition and locality in distributed computation. In: Proceedings of the 30th ACM Symposium on FOCS, pp. 364–369. ACM Press, New York (1989)

[BMW94] Bodlaender, H.L., Moran, S., Warmuth, M.K.: The distributed bit complexity of the ring: from the anonymous case to the non-anonymous case. Information and computation 114(2), 34–50 (1994)

[DMR08] Dinitz, Y., Moran, S., Rajsbaum, S.: Bit complexity of breaking and achieving symmetry in chains and rings. Journal of the ACM 55(1) (2008)

[Gho06] Ghosh, S.: Distributed systems - An algorithmic approach. CRC Press, Boca Raton (2006)

[KMNW05] Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 273–287. Springer, Heidelberg (2005)

[KMW04] Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proceedings of the 24 Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 300–309 (2004)

[KN99] Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press, Cambridge (1999)

[KOSS06] Kothapalli, K., Onus, M., Scheideler, C., Schindelhauer, C.: Distributed coloring in $o/spltilde/(/splradic/(logn))$ bit rounds. In: Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece, April 25-29. IEEE, Los Alamitos (2006)

[KW84] Karp, R.M., Widgerson, A.: A fast parallel algorithm for the maximal independent set problem. In: Proceedings of the 16th ACM Symposium on Theory of computing (STOC), pp. 266–272. ACM Press, New York (1984)

[KW06] Kuhn, F., Wattenhofer, R.: On the complexity of distributed graph coloring. In: Proceedings of the 25 Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 7–15. ACM Press, New York (2006)

[Lin92] Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. 21, 193–201 (1992)

[Lub86] Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. 15, 1036–1053 (1986)

[Lyn96] Lynch, N.A.: Distributed algorithms. Morgan Kaufmann, San Francisco (1996)

[MW05] Moscibroda, T., Wattenhofer, R.: Maximal independent set in radio networks. In: Proceedings of the 25 Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 148–157. ACM Press, New York (2005)

[NS95] Naor, M., Stockmeyer, L.J.: What can be computed locally? SIAM J. Comput. 24(6), 1259–1277 (1995)

[Pel00] Peleg, D.: Distributed computing - A Locality-sensitive approach. SIAM Monographs on discrete mathematics and applications (2000)

[San07] Santoro, N.: Design and analysis of distributed algorithms. Wiley, Chichester (2007)

[Tel00] Tel, G.: Introduction to distributed algorithms. Cambridge University Press, Cambridge (2000)

[Wat07] Wattenhofer, R. (2007), http://dcg.ethz.ch/lectures/fs08/distcomp/lecture/chapter4.pdf

[Yao79] Yao, A.C.: Some complexity questions related to distributed computing. In: Proceedings of the 11th ACM Symposium on Theory of computing (STOC), pp. 209–213. ACM Press, New York (1979)

Author Index