Andrzej Pelc
Michel Raynal (Eds.)

LNCS 3499

Structural Information and Communication Complexity

12th International Colloquium, SIROCCO 2005
Mont Saint-Michel, France, May 2005
Proceedings

Springer

Lecture Notes in Computer Science 3499

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

Editorial Board

Andrzej Pelc Michel Raynal (Eds.)

Structural Information and Communication Complexity

12th International Colloquium, SIROCCO 2005
Mont Saint-Michel, France, May 24-26, 2005
Proceedings

Springer

Volume Editors

Andrzej Pelc
Université du Québec en Outaouais
Département d'informatique
Gatineau, Québec J8X 3X7, Canada
E-mail: Andrzej.Pelc@uqo.ca

Michel Raynal
Université de Rennes 1, IRISA
Campus de Beaulieu, Avenue du General Leclerc, 35042 Rennes, France
E-mail: raynal@irisa.fr

Library of Congress Control Number: Applied for

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

ISSN 0302-9743
ISBN-10 3-540-26052-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-26052-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11429647 06/3142 5 4 3 2 1 0

Preface

The Colloquium on Structural Information and Communication Complexity (SIROCCO) is an annual meeting focused on the relationship between algorithmic aspects of computing and communication. Over its 12 years of existence, SIROCCO has become an acknowledged forum bringing together specialists interested in the fundamental principles underlying interplay between information, communication and computing.

SIROCCO 2005 was the twelfth in this series, held in Mont Saint-Michel, France, May 24–26, 2005. Previous SIROCCO colloquia took place in Ottawa (1994), Olympia (1995), Siena (1996), Ascona (1997), Amalfi (1998), Lacanau-Océan (1999), L'Aquila (2000), Val de Nuria (2001), Andros (2002), Umeå (2003) and Smolenice Castle (2004).

SIROCCO covers topics such as distributed and parallel computing, information dissemination, communication complexity, interconnection networks, high-speed networks, wireless networks, mobile computing, optical computing, and related areas.

The 48 contributions submitted to SIROCCO 2005 were subject to a thorough refereeing process and 22 high-quality submissions were selected for publication. We thank the Program Committee members for their excellent and careful work. Our gratitude extends to the numerous subreferees for their valuable refereeing. We also acknowledge the effort of all authors who submitted their contributions.

We thank the invited speakers at this colloquium, Amotz Bar-Noy (New York) and Cyril Gavoille (Bordeaux) for accepting our invitation to share their insights on new developments in their areas of interest. Amotz Bar-Noy delivered a talk "*Cellular Networks: Where Are the Mobile Users?*" and Cyril Gavoille presented "*Distributed Data Structures: a Survey.*"

We would like to express our sincere gratitude to the Steering Committee chair David Peleg (Rehovot) for his enthusiasm and invaluable consultations, and to the organizing team chaired by Elisabeth Lebret from INRIA. Special thanks are due to Rastislav Královič who, besides being a Program Committee member, helped us a lot with all technical work. Finally, we would like to thank IRISA, INRIA, the Université de Rennes 1 and the "Fondation Michel Metivier" for their support.

May 2005 Andrzej Pelc and Michel Raynal

Organization

Program Committee

Carole Delporte-Gallet (Paris, France)
Paola Flocchini (Ottawa, Canada)
Leszek Gąsieniec (Liverpool, UK)
Rachid Guerraoui (Lausanne, Switzerland)
Dariusz Kowalski (Warsaw, Poland)
Rastislav Královič (Bratislava, Slovakia)
Danny Krizanc (Middletown, CT, USA)
Shay Kutten (Haifa, Israel)
Bernard Mans (Macquarie, Australia)
Toshimitzu Masuzawa (Osaka, Japan)
Achour Mostefaoui (Rennes, France)
Jaroslav Opatrny (Montreal, Canada)
Andrzej Pelc, **Co-chair** (Gatineau, Canada)
Sergio Rajsbaum (UNAM, Mexico)
Michel Raynal, **Co-chair** (IRISA, Rennes, France)
Alex Shvartsman (Storrs, CT, USA)
Ugo Vaccaro (Salerno, Italy)

Organizing Committee

Elisabeth Lebret (Chair, INRIA)
Lydie Mabil (INRIA)
Florence Santoro (INRIA)

Webmaster and publication chair: Rastislav Královič (Bratislava, Slovakia)

Steering Committee Chair

David Peleg (Rehovot, Israel)

Referees

Christoph Ambuhl
Surender Baswana
Paolo Boldi
Marek Chrobak
Paolo D'Arco
Shantanu Das
Gianluca De Marco
Xavier Defago
Rafal Dowgird
Pavol Ďuriš
Stefan Dziembowski
Thomas Erlebach
Hugues Fauconnier
Sandor Fekete
Guillaume Fertin
Alexey Fishkin
Satoshi Fujita
Jana Gajdošiková
Clemente Galdi
Anna Gambin
Cyril Gavoille
Franciszek Grzegorek
Jaime Gutierrez
Taisuke Izumi
Hirotsugu Kakugawa
George Karakostas
Branislav Katreniak
Richard Královič
Lukasz Kowalik
Maciej Kurowski
Bruce Litow
Adam Malinowski
David Manlove
Giovanna Melideo
Marcin Mucha
Yoshihiro Nakaminami
Fukuhito Ooshita
Martin Pál
Dana Pardubská
Jakub Pawlewicz
Franck Petit
Giuseppe Prencipe
Andrzej Proskurowski
André Raspaud
Adele Rescigno
Piotr Sankowski
Paul Sant
Nicola Santoro
Mordechai Shalom
Sunil Shende
Samia Soussi
Lukasz Sznuk
Vincent Villain
Ivan Visconti
Koichi Wada
Prudence Wong

Table of Contents

Abstracts of Invited Talks 1

On Designing Truthful Mechanisms for Online Scheduling
Vincenzo Auletta, Roberto De Prisco, Paolo Penna, Giuseppe Persiano 3

On Private Computation in Incomplete Networks
Amos Beimel 18

Traffic Grooming on the Path
Jean-Claude Bermond, Laurent Braud, David Coudert 34

Range Augmentation Problems in Static Ad-Hoc Wireless Networks
Davide Bilò, Guido Proietti 49

On the Approximability of the $L(h,k)$-Labelling Problem on Bipartite Graphs
Tiziana Calamoneri, Paola Vocca 65

A Tight Bound for Online Coloring of Disk Graphs
Ioannis Caragiannis, Aleksei V. Fishkin, Christos Kaklamanis, Evi Papaioannou 78

Divide and Conquer Is Almost Optimal for the Bounded-Hop MST Problem on Random Euclidean Instances
Andrea E.F. Clementi, Miriam Di Ianni, Angelo Monti, Massimo Lauria, Gianlucca Rossi, Riccardo Silvestri 89

Distributed Exploration of an Unknown Graph
Shantanu Das, Paola Flocchini, Amiya Nayak, Nicola Santoro 99

Two Absolute Bounds for Distributed Bit Complexity
Yefim Dinitz, Noam Solomon 115

Finding Short Right-Hand-on-the-Wall Walks in Graphs
Stefan Dobrev, Jesper Jansson, Kunihiko Sadakane, Wing-Kin Sung 127

Space Lower Bounds for Graph Exploration via Reduced Automata
Pierre Fraigniaud, David Ilcinkas, Sergio Rajsbaum, Sébastien Tixeuil 140

Communications in Unknown Networks: Preserving the Secret of Topology
Markus Hinkelmann, Andreas Jakoby 155

An Improved Algorithm for Adaptive Condition-Based Consensus
Taisuke Izumi, Toshimitsu Masuzawa 170

Biangular Circle Formation by Asynchronous Mobile Robots
Branislav Katreniak 185

Hardness and Approximation Results for Black Hole Search in Arbitrary Graphs
Ralf Klasing, Euripides Markou, Tomasz Radzik, Fabiano Sarracco 200

On Semi-perfect 1-Factorizations
Rastislav Královič, Richard Královič 216

Free-Riders in Steiner Tree Cost-Sharing Games
Paolo Penna, Carmine Ventre 231

On the Feasibility of Gathering by Autonomous Mobile Robots
Giuseppe Prencipe 246

Majority and Unanimity in Synchronous Networks with Ubiquitous Dynamic Faults
Nicola Santoro, Peter Widmayer 262

Minimizing the Number of ADMs in SONET Rings with Maximum Throughput
Mordechai Shalom, Shmuel Zaks 277

Optimal Gossiping in Square Meshes in All-Port Mode and with Short Packets
Rui Wang, Francis C.M. Lau 292

Geometric Routing Without Geometry
Mirjam Wattenhofer, Roger Wattenhofer, Peter Widmayer 307

Author Index 323

Cellular Networks: Where Are the Mobile Users?

(Invited Talk)

Amotz Bar-Noy

CUNY, New York

Abstract. Mobiles are roaming in a cellular network. Unless they report their new location each time they cross boundaries of cells, the system must conduct a search operation to find their exact location. Reporting new locations by mobiles consumes expensive up-link communication lines. Therefore, in current and future cellular networks, at each point in time for any particular mobile, the system knows only a zone of cells containing the one cell which is the location of this mobile. For this zone, the system maintains a profile that predicts the exact location of the mobile by associating a probability with each cell in the zone. An efficient search should optimize usage of down-link communication lines and the time needed to find the mobile.

This model gives rise to many optimization problems. This talk discusses some of them. We first describe the optimal dynamic programming solution that finds a mobile that is located in a zone of n cells in no more than D rounds. This solution assumes an a priori knowledge of the mobile's profile. We then present solutions in which the system develops a mobile's profile while searching for that mobile more than once. The above solutions are for locating one mobile. Next, we address search operations involving m mobiles where m can be greater than one. One example is the call conference search in which the system must find all the m mobiles. Another example is the yellow pages search where the search is over once one out of the m mobiles is found. Finding an optimal solution to the conference call problem is NP-hard. We therefore present an efficient approximation solution. For the yellow pages problem we discuss work in progress. We conclude with the privacy issue by exploring the tradeoff between the accuracy of the profiles and the efficiency of the optimal solutions that are based on these profiles.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, p. 1, 2005.

Distributed Data Structures: A Survey

(Invited Talk)

Cyril Gavoille

LABRI, Bordeaux

Abstract. This survey concerns the role of data structures for compactly storing and representing various types of information in a localized and distributed fashion. Traditional approaches to data representation are based on global data structures, which require access to the entire structure even if the sought information involves only a small and local set of entities. In contrast, localized data representation schemes are based on breaking the information into small local pieces, or *labels*, selected in a way that allows one to infer information regarding a small set of entities directly from their labels, without using any additional (global) information.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, p. 2, 2005.

On Designing Truthful Mechanisms for Online Scheduling*

Vincenzo Auletta[1], Roberto De Prisco[1,2], and Paolo Penna[1], and Giuseppe Persiano[1]

[1] Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno, via **S. Allende** 2, I-84081 Baronissi (SA), Italy
{auletta, robdep, penna, giuper}@dia.unisa.it
[2] Faculty Group at Akamai Technologies, Cambridge, MA, USA

Abstract. We study the *online* version of the scheduling problem involving *selfish agents* considered by Archer and Tardos [FOCS 2001]: jobs must be scheduled on m parallel related machines, each of them owned by a different *selfish agent*.

Our study focuses on general techniques to translate approximation/competitive algorithms into equivalent approximation/competitive *truthful mechanisms*. Our results show that this translation is more problematic in the online setting than in the offline one. For $m = 2$, we develop an offline and an online "translation" technique which, given *any* ρ-approximation/competitive (polynomial-time) algorithm, yields an $f(\rho)$-approximation/competitive (polynomial-time) mechanism, with $f(\rho) = \rho(1 + \varepsilon)$ in the offline case, for every $\varepsilon > 0$. By contrast, one of our lower bounds implies that, in general, online ρ-competitive algorithms cannot be turned into $\rho(1+\varepsilon)$-competitive mechanisms, for some $\varepsilon > 0$ and every $m \geq 2$.

We also investigate the issue of designing new online algorithms from scratch so to obtain efficient competitive mechanisms, and prove some lower bounds on a class of "natural" algorithms. Finally, we consider the variant introduced by Nisan and Ronen [STOC 1999] in which machines can be *verified*. For this model, we give a $O(1)$-competitive online mechanism for *any* number of machines and prove that some of the above lower bounds can be broken.

1 Introduction

Optimization problems dealing with resource allocation are classical algorithmic problems and they have been studied for decades in several models. Typically, algorithms are evaluated by comparing the (measure of) the solutions they return to the best possible one. In particular, one tries to estimate the loss of

* Work supported by the European Project IST-2001-33135, Critical Resource Sharing for Cooperation in Complex Systems (CRESCCO).

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 3–17, 2005.

performance due to the lack of computational resources (*approximation ratio*) or to the lack of information (*competitive ratio*).

In both settings, the underlying hypothesis is that the input is (eventually) available to the algorithm (either from the beginning in off-line algorithms or during its execution in on-line algorithms). This assumption cannot be considered realistic in the context of modern networks like the Internet where certain information regarding the resources are not directly available to the "protocol". Indeed, since the resources are owned/controlled/used by different *self-interested* entities (e.g., corporations, autonomous systems, etc.). Each of these entities, or *selfish agents*, hold some *private information* which is needed in order to compute an optimal resource allocation (e.g., routing the traffic over the Internet requires routers of different autonomous systems to exchange information on which routers can process traffic faster). Each agent can possibly *misreport* his/her piece of information if this leads the system to compute a solution that is more beneficial for him/her. This, in spite of the fact that such a solution may *not* be not globally optimal.

The field of *Mechanism Design* is the branch of Game Theory and Microeconomics that studies how to design complex auctions, also termed *mechanisms*, which guarantee that no agents has an incentive in misreporting his/her piece of information. Loosely speaking, a mechanism is a pair $M = (A, P)$, where A is an algorithm computing a solution, and $P = (P^1, \ldots, P^n)$ is the vector of payment functions (see Sect. 1.1 for a formal definition). Selfish agents are suppose to be rational and thus will deviate from the truth-telling strategy (in our problem, to report $r_i = s_i$) only if a better one exists. Therefore, one seeks for *truthful* mechanisms, that is, mechanisms that guarantee that every agent i can maximize his/her net profit or *utility* by playing the truth-telling strategy (see Sect. 1.1).

In this work we consider the *online* version of a basic scheduling/routing problem involving *selfish agents*, first addressed by Archer and Tardos [2]. We will investigate the approximation/competitive ratio of truthful mechanisms for this problem. Our goal is to quantify the (further) loss of optimality due to the combination of selfish agents with the online setting. Central to our study is the existence of general techniques that allow to translate ρ-approximation/competitive algorithms into a $f(\rho)$-approximation/online mechanisms, for some function $f(\cdot)$.

1.1 The Problem

Offline Selfish Version. Consider the problem of scheduling jobs on related machines ($\mathrm{Q}||\mathrm{C}_{\max}$): We are given a set of m machines with speed $s_1, s_2, \ldots, s_m$ and a set of n jobs of size $J_1, J_2, \ldots, J_n$. We want to assign every job to a machine so to minimize the *makespan*, that is, the maximum over all machines of w_i/s_i, where w_i is the sum of the job weights assigned to machine i. When the set of machines m is fixed, this problem version is commonly denoted to as $\mathrm{Q}_m||\mathrm{C}_{\max}$.

We study the selfish version of the $Q||C_{\max}$ problem in which each machine i is owned by a selfish agent and the corresponding speed s_i is known to that agent only. In particular, any schedule S that assigns load w_i to machine i is valuated by agent i as $v^i(S)$, where

$$v^i(S) \stackrel{\text{def}}{=} -w_i/s_i,$$

that is, the opposite of the completion time of machine i. Intuitively, $v^i(S)$ represents how much user i likes solution S. This model has been first considered by Archer and Tardos [2].

We stress that our goal is to compute a solution S which minimizes the makespan with respect to the *true* machine speeds $s_1, \ldots, s_m$. Hence, we need to provide some incentive (e.g., a payment P^i) to the each agent i in order to let him/her truthfully report his/her speed. Formally, a *mechanism* is a pair $M = (A, P_A)$, where $P_A = (P_A^1, \ldots, P_A^m)$, and A is a scheduling algorithm. Each agent i reports its type b_i which is not necessarily the true type $t_i \stackrel{\text{def}}{=} 1/s_i$. Algorithm A gets in input the reported types $b = (b_1, \ldots, b_m)$, and each agent i receives a payment equal to $P_A^i(b, J)$. Obviously, each agent i wants to maximize the resulting net profit or *utility* defined as

$$u_i^M(b, J) \stackrel{\text{def}}{=} P_A^i(b, J) + v^i(A(b, J)).$$

Each agent *knows* both algorithm A and the payment function P_A^i.

A mechanism is said to be *truthful with dominant strategies* (or simply *truthful*) if the payments P_A and the algorithm A guarantee that no agent obtains a larger utility when reporting $b_i \neq t_i$, independently of the other agents' reported types; that is, for all J, for all reported types $b_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_m)$ of all the agents except i, and for all possible declarations b_i of agent i, it holds that

$$u_i^M((t_i, b_{-i}), J) \geq u_i^M((b_i, b_{-i}), J),$$

where the writing (x, b_{-i}) denotes the vector $(b_1, \ldots, b_{i-1}, x, b_{i+1}, \ldots, b_m)$. We stress that no agent i has any advantage from knowing the true speeds t_{-i} of the other agents: indeed, the utility of agent i does *not* depend on the speeds of the other agents (the work/payment assigned to machine/agent i depend on the agent bids b only). If M guarantees that the utility is non-negative for all agents i that report their true type, then we say that the mechanism enjoys the *voluntary participation* property.

Online Selfish Version. In the *online* version of $Q||C_{\max}$, jobs arrive one-by-one and must be scheduled upon their arrival. Moreover, jobs cannot be reallocated. For any (possibly infinite) sequence of jobs $J = J_1 J_2 \cdots$, we let J^k denote the prefix $J_1 J_2 \cdots J_k$ of the first k jobs, for $1 \leq k \leq |J|$. Before any job appears, each agent declares her type and we denote by $b = (b_1, \ldots, b_m)$ the vector of declared types. An *online mechanism* for $Q||C_{\max}$ is a pair $M = (A, P)$ where P is a sequence of payment functions P_i^k, for $i = 1, \ldots, m$ and $k > 0$ such that

- The algorithm A is an online algorithm for $Q||C_{max}$; we denote by $w_i^A(b, J^k)$ the sum of the job sizes assigned to machine i by the solution computed by A on input J^k and vector b of declared types.
- When the k-th jobs arrives, it is assigned by A to a machine and each agent i receives *non-negative* payment $P_i^k(b, J^k)$. That is, we are not allowed to ask money back from the agents.

The total payment received by agent i after k jobs is equal to $P_i(b, J^k) = \sum_{j=1}^{k} P_i^j(b, J^j)$.

Definition 1 (online truthful mechanism). *We say that an online mechanism is* truthful *with respect to* dominant strategies *if for any prefix J^k of J, for all b_{-i}, and for all types t_i, the function $u_i^M((b_i, b_{-i}), J^k)$ is maximized for $b_i = t_i$.*

Verifiable Machines. We also study the online version of the model proposed by Nisan and Ronen [9] of verifiable machines. Here the payment for each job is awarded after the job is released by the machine (we stress that a machine cannot release a job assigned to it before the job has been executed). Intuitively, if a machine has received positive load, the mechanism can verify whether the machine lied declaring to be *faster* and, if so, the machine receives no payment.

1.2 Previous Results

Archer and Tardos [2] have characterized the (offline) algorithms A for $Q||C_{max}$ for which there exist payment functions P such that (A, P) is a truthful mechanism. In particular they show that if an algorithm A is monotone (that is, it satisfies $w_i^A((b_i', b_{-i}), J) \leq w_i^A((b_i, b_{-i}), J)$, for all $b_i' > b_i$) then there exists a payment function P such that (A, P) is truthful. Under mild assumptions on A, it is possible to define the payment function to guarantee voluntary participation. They also gave a monotone optimal (exponential-time) algorithm for $Q||C_{max}$ and a $(3+\varepsilon)$-approximate randomized (polynomial-time) monotone algorithm. In [4] we gave a $(4+\varepsilon)$-approximate deterministic (polynomial-time) monotone algorithm for $Q_m||C_{max}$. Recently and independently from this work, Andelman *et al* [1] provided an elegant technique for turning any ρ-approximation algorithm for $Q_m||C_{max}$ into a $\rho(1+\varepsilon)$-approximation monotone mechanism. As a result, given any polynomial-time $(1+\varepsilon)$-approximation algorithm for this problem, one can obtain a $(1+\varepsilon)$-approximation mechanism running in polynomial time. They indeed settle the approximation guarantee of the $Q_m||C_{max}$ by obtaining a fully polynomial-time approximation scheme which is monotone. Moreover, they provide a 5-approximation truthful mechanism for the $Q||C_{max}$ problem, i.e., for any number of machines.

Nisan and Ronen [9] considered the case of unrelated machines and gave a randomized 7/4-approximate truthful mechanism for two machines and a deterministic m-approximate truthful mechanism for any number of machines. Moreover, they proved that no deterministic truthful mechanism can be $(2-\varepsilon)$-

approximate for $m \geq 2$ machines. Nisan and Ronen also considered the case of verifiable unrelated machines and gave a polynomial-time $(1+\varepsilon)$-approximate truthful mechanism for any fixed number of machines. For the case of verifiable related machines (that is $Q||C_{\max}$), in [6], we characterized the algorithms A for which there exist payment functions P such that (A, P) is a truthful mechanism. Based on this we developed a polynomial-time $(1+\varepsilon)$-approximate truthful mechanism for the offline version of $Q||C_{\max}$.

1.3 Our Contribution

A central question in (algorithmic) mechanism design is to translate approximation/online algorithms into approximation/online mechanisms: given an algorithm A of approximation/competitive ratio ρ, can we obtain a monotone algorithm $\overline{A}$ with the same approximation/competitive ratio? A general approach to the design of approximation/competitive mechanisms might be that of developing general "monotonization" techniques: starting from any ρ-approximation/competitive algorithm A, transform A into a monotone algorithm $\overline{A}$ with approximation/competitive ratio $\overline{\rho}$ depending on ρ. We first consider the $Q_2||C_{\max}$ problem for which we provide the following two general results:

Offline Case: Every polynomial-time ρ-approximation algorithm can be transformed into a *monotone* polynomial-time $(\rho+\varepsilon)$-approximation algorithm $\overline{A}$, for every $\varepsilon > 0$ (Theorem 3). This result is a special case of the one obtained independently by Andelman *et al* [1]: indeed, their monotonization technique extends our result to any fixed number of machines.

Online Case: Given an online ρ-competitive algorithm A, for every $t > 0$, it is possible to obtain an online monotone algorithm $\overline{A}_t$ whose competitive ratio $\overline{\rho}$ satisfies $\overline{\rho} \leq \max\{\rho \cdot t, 1 + 1/t\}$ (Theorem 5). Moreover, the same bound holds if A is a ρ-competitive algorithm (only) for *identical speeds*. The "monotonization" of the greedy algorithm [1] thus yields and online mechanism whose competitive ratio is at most $1 + \sqrt{7}/2 < 1.823$ (Corollary 2).

It is natural to ask whether the loss of performance due to our "monotonization" for the online setting is really necessary, and whether (some of the) existing algorithms could preserve their competitive guarantee (after being turned into a monotone one).

We first show a general lower bound on *monotone online* algorithms. Consider the problem restricted to instances for which $s_{\max}/s_{\min} = r$, for any $r > 0$. Then, no such algorithm can be less than $\rho(r)$-competitive, with $\rho(r) \geq \min\{r, 1+1/r\}$ (Theorem 6). This gives a general lower bound of $\phi \simeq 1.62$, which also holds for sequences of two jobs (Corollary 3). At least for such sequences our technique is optimal: indeed, since the greedy algorithm is 1-competitive, our method yields a ϕ-competitive online algorithm (simply choose $t = \phi$).

[1] This algorithm, also known in the literature as ListScheduling, assigns the current job J_i to the machine that minimize the completion time of J_i.

An underlying implicit assumption in designing scheduling algorithms is that, for the same set of jobs, speed vectors $s = (s_1, s_2)$ and $s_\alpha = (\alpha s_2, \alpha s_1)$ lead to the same solution (modulo a machine re-indexing). We show that this (apparently natural) way of proceeding must necessarily lead to online monotone algorithms whose competitive ratio is not smaller than 2. In particular, we isolate two pathological facts that, each of them alone, prevent from having a non-trivial competitive ratio (see Theorems 7-8): (i) the first job is always assigned to the fastest machine, and (ii) solution for (s_1, s_2) is isomorphic (modulo a index exchange) to that for (s_2, s_1).

It is worth observing that the *lack of information* plays a central role both in the *online* and in the *selfish* setting of the problem. In the online setting we do not know the "future;" when dealing with "selfish" agents we do not know part of the input. Our results (see Table 1) show that the combination "online+selfish" makes the $Q_2||C_{\max}$ problem harder than both the offline with selfish agents and the online (without selfish agents) versions. In particular, for $\sqrt{2} < r \leq \phi$, it holds that (i) r is a lower bound for any online monotone algorithm (i.e., any mechanism), while (ii) there is an upper bound $\rho \leq 1 + 1/(r+1) < r$ provided by the greedy for the online case (without selfish agents).

Table 1. Our and previous results for the case of two machines: all lower bounds also apply to exponential-time algorithms, while upper bounds are obtained via polynomial-time ones

	Offline		**Online**	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Non Selfish	1 [trivial]	$1+\varepsilon$ [8]	$1+1/(r+1)$ [folklore]	$1+1/(r+1)$, for $r \leq \phi$ [3–greedy] $1+1/r$, for $r > \phi$ [3–greedy]
Selfish	1 [trivial]	$1+\varepsilon$ [Cor. 1] or [1]	$\min\{r, 1+1/r\}$ [Thm. 6]	$1+\sqrt{7}/2 < 1.823$ [Thm. 5 and Cor. 2]

All our lower bounds also apply to $Q_m||C_{\max}$, for any $m > 2$. As for the upper bounds, in Sect. 6 we present a 12-competitive algorithm for *any number* of *verifiable* machines. This is the first constant-competitive truthful online algorithm for any number of machines ($Q||C_{\max}$).

The ability to "verify" machines has been proved to yield better approximation mechanisms in the offline case for other scheduling problems [9, 6]. The results here show that the same happens also for the online version of $Q_m||C_{\max}$, for any $m \geq 2$. By contrast, the results by Andelman *et al* [1] imply that in the *offline* setting verification does not help for $Q_m||C_{\max}$, for any $m \geq 2$.

Due to lack of space, some of the proofs are omitted in this extended abstract. We refer the interested reader to the full version of this work [5].

Notation. Throughout the paper s_i will denote the speed of the i-th machine, t_i its type (i.e., $t_i = 1/s_i$) and b_i the type reported by agent i.

2 Characterization of Online Truthful Mechanisms

For the offline case, Archer and Tardos [2] characterized the class of algorithms that can be used as part of a truthful mechanism. More precisely, we have the following definition and theorem.

Definition 2 (monotone algorithm). *An algorithm A is* monotone *if, for every i, for every J, for every b_{-i}, for every b_i and $b'_i > b_i$ it holds that*

$$w_i^A((b'_i, b_{-i}), J) \leq w_i^A((b_i, b_{-i}), J),$$

where $w_i^A((b_i, b_{-i}), J)$ is the load assigned to machine i when J is the job sequence and agents report types (b_i, b_{-i}).

Theorem 1 (offline characterization [2]). *A mechanism $M = (A, P)$ is truthful if and only if A is monotone. Moreover, for every monotone algorithm A, there exist payment functions P such that (A, P) is truthful and satisfies voluntary participation if and only if $\int_0^\infty w_i^A((u, b_{-i}), J)\, du < \infty$ for all i, J, and b_{-i}. In this case, we can take the payments to be*

$$P_i((b_i, b_{-i}), J) = b_i \cdot w_i^A((b_i, b_{-i}), J) + \int_{b_i}^{\infty} w_i^A((u, b_{-i}), J) du. \tag{1}$$

Next, we translate the result above into the online setting. We will use the characterization to obtain our upper and lower bounds.

Theorem 2 (online characterization). *An online mechanism $M = (A, P)$ is truthful if and only if A is an online monotone algorithm. Moreover, for every online monotone algorithm A, there exists a payment function P such that (A, P) is truthful. In addition, there exist payment functions P_i^k such that $P_i^k((b_i, b_{-i}), J^k) \geq 0$ for all J, k and (b_i, b_{-i}).*

3 Monotonization Techniques

3.1 Offline Monotonization

In this section we give a general technique for transforming any ρ-approximate algorithm A for $\mathrm{Q}_2||\mathrm{C}_{\max}$ into an offline $(\rho+\varepsilon)$-approximate *monotone* algorithm $\overline{A}$. Essentially, our monotonization technique goes thorough two steps: (i) we first consider an algorithm A_γ which is noting but A running over speeds rounded to the closest power of γ, and (ii) we inspect the solutions of A_γ by varying only one of the two machine speeds over a polynomial number of values: indeed, considering only instances $(1, \gamma^i)$ will guarantee the monotonicity.

In the sequel we let A be any algorithm satisfying the following two properties:

$$w_1^A((s_{\min}, s_{\max}), J) \leq w_2^A((s_{\min}, s_{\max}), J), \tag{2}$$

$$A((s_{\min}, s_{\max}), J) = A((1, s_{\max}/s_{\min}), J). \tag{3}$$

This is without loss of generality since any offline algorithm which violates any of the two conditions above can be easily modified without any loss in the approximation guarantee.

Theorem 3. *For algorithm A and every $\varepsilon > 0$, there exists a monotone algorithm $\overline{A}$ such that, if A is a (polynomial-time) ρ-approximation algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$, then algorithm $\overline{A}$ is a monotone (polynomial-time) $(\rho + \varepsilon)$-approximation algorithm.*

Corollary 1. *For every $\varepsilon > 0$, there exists a polynomial-time $(1 + \varepsilon)$-approximation mechanism for $\mathrm{Q}_2||\mathrm{C}_{\max}$.*

Remark 1. Recently and independently from this work, the above result has been improved in [1]. The authors provided a more general technique for obtaining a monotone algorithm $\overline{A}$ for the $\mathrm{Q}_m||\mathrm{C}_{\max}$ problem. In particular, Corollary 1 can be improved so to obtain a monotone FPTAS for this problem version. Moreover, for the case $m = 2$, their technique essentially leads to the same algorithm as the one proposed here.

3.2 Online Monotonization

The basic idea is to output a "fixed" allocation that ignores the machine speeds as long as they are "almost the same": this allocation is based on the machine indexes only. As soon as one machine becomes significantly faster than the other, we assign all jobs to that machine. The algorithm template in Figure 1 implements this idea.

Algorithm A-asymmetric
1. fix a threshold $t > 1$;
2. $s_{\max} := \max\{s_1, s_2\}$; $s_{\min} := \min\{s_1, s_2\}$;
3. if $s_{\max}/s_{\min} \leq t$ then
   run online algorithm A on machine speeds $s'_1 = s'_2 = 1$;
1. else assign every job to machine of speed $s_{\max}$;

Fig. 1. An online monotone algorithm for two machines

Theorem 4. *For every $t > 1$ and for every online algorithm A for $\mathrm{Q}_2||\mathrm{C}_{\max}$, algorithm A-asymmetric is an online* monotone *algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$.*

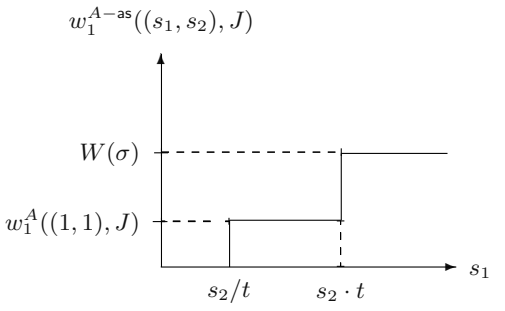


Fig. 2. The work curve $w_1^{A-\mathsf{as}}((\cdot, s_2), J)$ of algorithm A-asymmetric

Proof. The algorithm A-asymmetric is clearly an online algorithm since the choice of which strategy to use is done based on the machine speeds, which do not change during the online phase (i.e., when jobs arrive).

Let $w_i^{A-\mathsf{as}}((s_1, s_2), J)$ denote the work assigned to machine i by A-asymmetric on input J and speeds (s_1, s_2), for $i = 1, 2$. Also let $W(J) = \sum_{a=1}^{|J|} J_a$. Observe that, by definition of A-asymmetric, we have that

$$w_1^{A-\mathsf{as}}((s_1,s_2),J) = \begin{cases} w_1^A((1,1),J) & \text{if } s_1 \le s_2 \text{ and } s_1 \ge s_2/t, \\ w_1^A((1,1),J) & \text{if } s_1 > s_2 \text{ and } s_1 \le s_2 \cdot t, \\ 0 & \text{if } s_1 \le s_2 \text{ and } s_1 < s_2/t, \\ W(J) & \text{if } s_1 \ge s_2 \text{ and } s_1 > s_2 \cdot t. \end{cases} \tag{4}$$

Notice that, since $t > 1$, we have $s_2/t < s_2$. From the above equation we obtain the allocation curve in Figure 2, which clearly implies the monotonicity w.r.t. machine 1.

By using the same argument, we can prove the monotonicity of the function $w_2^{A-\mathsf{as}}((s_1, \cdot), J)$. This completes the proof.

Theorem 5. *For every ρ-competitive online algorithm A for $\mathrm{Q}_2||\mathrm{C}_{\max}$, and for every $t > 1$, algorithm A-asymmetric is ρ^{as}-competitive algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$ for $\rho^{as} = \max\{\rho \cdot t, 1 + 1/t\}$.*

Corollary 2. *There exists an online monotone algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$ whose competitive ratio is $\frac{1+\sqrt{7}}{2} \simeq 1.823$.*

Proof. Let us consider the greedy algorithm A_{gr} whose competitive ratio on two machines of identical speed is $3/2$ [8]. Then, from Theorem 5 we have that algorithm A_{gr}-asymmetric has competitive-ratio bounded from above by $\max\{3t/2, 1 + 1/t\}$. We minimize this quantity by choosing $t > 1$ such that $3t/2 = 1 + 1/t$. This corresponds to $t = \frac{1+\sqrt{7}}{3}$, thus yielding a competitive ratio equal to $\frac{1+\sqrt{7}}{2} \simeq 1.823$.

4 Lower Bound for Online Selfish Scheduling

In this section, we provide a general lower bound for online $Q||C_{\max}$ with selfish agents. This result proves that the selfish online version of this problem is more difficult than the corresponding version of the problem with no selfish agents, even for two machines.

Theorem 6. *For every $m \geq 2$ and every $r > 1$, no monotone online algorithm can be less than ρ_r-competitive, where $\rho_r = \min\{r, 1 + 1/r\}$. This holds even for two jobs.*

Proof. By contradiction, let A be an online monotone ρ-competitive algorithm on m machines, for $\rho < \min\{r, 1 + 1/r\}$. Let $J = (J_1, J_2) = (1, r)$ and let $s = (1, \ldots, 1)$. Observe that $A(s, J)$ cannot allocate two jobs on the same machine otherwise A would produce a solution of cost $1 + r$, while the optimum costs r, contradicting the hypothesis that A is ρ competitive. Without loss of generality, assume $w_1^A = 1$ and $w_2^A = r$.

Suppose now that speed of machine 1 is increased to r. Since A is monotone also with respect to the sequence J^1, then it must be the case $w_1^A((r, s_{-1}), J^1) = w_1^A(s, J^1) = 1$. Since we do not allow jobs to be reassigned, we have to consider only two cases:

($w_1^A((r, s_{-1}), J) = 1 + r$.) In this case, $(1 + 1/r)/\mathsf{opt}((r, s_{-1}), J) = 1 + 1/r$, thus contradicting the hypothesis that A is ρ-competitive.

($w_1^A((r, s_{-1}), J) = 1$.) This gives $r/\mathsf{opt}((r, 1), J) = r$, contradicting the hypothesis that A is ρ-competitive.

Hence the theorem follows.

Corollary 3. *No monotone online algorithm for $Q_2||C_{\max}$ can be less than ϕ-competitive. This holds even for two jobs, in which case the bound is tight since there exists a ϕ-competitive online monotone algorithm.*

Proof. The lower bound follows from Theorem 6 by taking $r = \phi = 1+1/r$. As for the upper bound, consider algorithm A_{gr}-asymmetric with $t = \phi$. For sequences of two jobs A_{gr} is 1-competitive. Theorem 5 thus implies a competitive ratio $\rho \leq \max\{\phi, 1 + 1/\phi\} = \phi$.

5 On Building Online Monotone Algorithms

Apparently, a good way to obtain online monotone algorithms is to guarantee that faster machines receive more work. In particular, when dealing with the case of only one job, a natural (optimal) solution is to assign it to the fastest machine. This is also what a direct use of the so called *Vickery auction* [10] would give for our problem. (These so called "sealed bid" auctions compute a solution only based on the agents' bids – see e.g. [9, 2].) This motivates the following definition:

Definition 3 (best-first algorithm). *An algorithm A is* best-first *if the first job is always assigned to the fastest machine.*

In addition, it is natural to treat speeds (s_1, s_2) and $(\alpha s_2, \alpha s_1)$ as essentially the same instance: by rescaling, and reindexing machines we reduce both of them to $(1, s_2/s_1)$. Hence, the algorithm is supposed to produce the same solution. We thus consider the following class of algorithms:

Definition 4 (symmetric algorithm). *An algorithm A is* symmetric *if, for any two speed vectors s and s' such that, for a permutation π, $s' = \pi(s)$ it holds that, for all i, $w_i^A(s, J) = w_{\pi(i)}^A(s', J)$.*

A simple argument shows that any monotone algorithm which is best-first *and* symmetric cannot be less than 2-competitive, even for $m = 2$. There are, however, algorithms which are best-first though not symmetric or vice versa. Does any of these give a better performance? The next two results prove that the answer to this question is no.

Theorem 7. *For every $m \geq 2$, no online monotone best-first algorithm for $\mathrm{Q}_m||\mathrm{C}_{\max}$ can be better than 2-competitive. This holds even for two jobs.*

Proof. By contradiction, let A be a best-first, monotone and $(2-\gamma)$-competitive algorithm, for some $\gamma > 0$. Consider $J = (1, 1+\varepsilon)$, for some $\varepsilon > 0$, and let $s_1 = 1$, $s_2 = 1+\varepsilon$ and $s_i = \varepsilon$, with $3 \leq i \leq m$. Notice that, since A is $(2-\gamma)$-competitive and best-first, it is possible to take ε sufficiently small so that A assigns the first job to machine 2 and the second job to machine 1.

Suppose now that speed of machine 2 is reduced to $1 - \varepsilon$. We observe that A, on input J and $(1, 1 - \varepsilon)$ assigns no jobs to machine 2. In fact, since it is best-first, it assigns the first job to machine 1. Moreover, since it is monotone, it has to assign a load to machine 2 not greater than 1. Thus, also the second job is assigned to machine 1. However, this implies that the solution computed by A has cost $2 + \varepsilon$, while the optimum has cost $1 + \varepsilon$. For ε sufficiently small, this contradicts the hypothesis that A is $(2 - \gamma)$-competitive.

Theorem 8. *For every $m \geq 2$, no online monotone symmetric algorithm for $\mathrm{Q}_2||\mathrm{C}_{\max}$ can be less than 2-competitive. This holds even for two jobs.*

Proof. We prove the theorem for $m = 2$. The extension to $m > 2$ is straighforward. Let us assume by contraddiciton that A is a monotone, symmetric, and $(2 - \gamma)$-competitive algorithm, for some $0 < \gamma < 1$. Consider $J = (1, 1 + \varepsilon)$, for some $\varepsilon > 0$ and let $s_1 = 1$ and $s_2 = 1 + \varepsilon$. For sufficiently small ε, algorithm A cannot allocate two jobs on the same machine. We thus have two possible solutions for algorithm A:

solution	machine 1 $s_1 = 1$	machine 2 $s_2 = 1 + \varepsilon$
SOL_1	$1 + \varepsilon$	1
SOL_2	1	$1 + \varepsilon$

Let ε be such that $\frac{2}{1+\varepsilon} > 2-\gamma$, that is, $\varepsilon < \frac{\gamma}{2-\gamma}$. We distinguish two cases:

($A((1,1+\varepsilon),J) = SOL_1$.) By monotonicity of A, $w_2^A((1,1),J) \leq w_2^A((1,1+\varepsilon),J) = 1$. If $w_2^A((1,1),J) = 0$, then we have a solution of cost $2+\varepsilon$, thus implying that A must be at least $(2+\varepsilon)/(1+\varepsilon)$-competitive. For our choice of ε, this would contradict the hypothesis that A is $(2-\gamma)$-competitive. Thus, $A((1,1),J)$ must coincide with solution SOL_1. Again, by monotonicity, it must hold $w_1^A((1+\varepsilon,1),J) \geq w_1^A((1,1),J) = 1+\varepsilon$. This contradicts the hypothesis that A is symmetric: indeed, from Definition 4 it holds that $w_2^A((1,1+\varepsilon),J) = w_1^A((1+\varepsilon,1),J) = 1+\varepsilon$, thus implying $w_1^A((1,1+\varepsilon),J) = 1$.

($A((1,1+\varepsilon),J) = SOL_2$.) Let us consider the allocation produced by A w.r.t. the first job only, that is, $J^1 = J_1 = 1$. Observe that, since jobs cannot be reassigned, it must hold $w_1^A((1,1+\varepsilon),J) = 1 = w_1^A((1,1+\varepsilon),J_1)$. Since A must be monotone also w.r.t. J^1, it holds that $w_2^A((1,1),J_1) \leq w_2^A((1,1+\varepsilon),J_1) = 0$. This implies $w_1^A((1,1),J_1) = J_1 = 1$. By monotonicity, $w_1^A((1+\varepsilon,1),J_1) = 1$. When the second job arrives, algorithm A can assign it to one of the two machines. If $w_1^A((1+\varepsilon,1),J) = J_1 + J_2 = 2+\varepsilon$, then A cannot be $(2-\gamma)$-competitive because of our choice of ε. Therefore, it must be the case that $w_1^A((1+\varepsilon,1),J) = 1$ and $w_2^A((1+\varepsilon,1),J) = J_2 = 1+\varepsilon$. This contradicts the hypothesis that A is symmetric: indeed, we have $w_1^A((1,1+\varepsilon),J) = 1 \neq w_2^A((1+\varepsilon,1),J_1) = 1+\varepsilon$.

Remark 2. Observe that our monotonization technique for offline algorithms requires the algorithm to be "monotonized" to be *both* best-first and symmetric. Thus, we implicity require the resulting algorithm to be best-first and symmetric as well.

6 Online Mechanisms with Verification

In this section we consider online mechanisms with *verification* [6]: the payments to an agent can be provided after the corresponding machine terminates; in this case, the mechanism can compute the payments as a function of such finish time(s). In the online setting, once machine j releases a job J_i, the mechanism observes a release time $r(J_i)$. However, machine j could declare to be slower (i.e., $b_j > s_j$) and release all jobs accordingly (i.e., $r(J_i) = J_i/b_j$).

In [6] we provide a sufficient condition to design truthful mechanisms:

Definition 5 (weakly monotone algorithm [6]). *An algorithm A is* weakly monotone *if, for every job sequence J, for every i, for every s_{-i} it holds that*

$$w_i^A((s_i,s_{-i}),J) = 0 \Rightarrow \forall s_i' < s_i, w_i^A((s_i',s_{-i}),J) = 0.$$

We will make use of the following result:

Theorem 9 ([6]). *An algorithm A admits a payment function p such that $M = (A, p)$ is truthful for the case of verifiable machines if and only if A is weakly monotone.*

We first observe that the greedy algorithm is weakly monotone. Therefore, we have the following result on the "power" of verification for the $\mathrm{Q}_2||\mathrm{C}_{\max}$ problem:

Theorem 10. *Let us consider the $\mathrm{Q}_2||\mathrm{C}_{\max}$ problem. There exists two functions $UB_v(\cdot)$ and $LB(\cdot)$, such that (i) no truthful mechanism can be less than $LB(r)$-competitive if machines cannot be verified, (ii) there is an $UB_v(r)$-competitive truthful mechanism for the case of verifiable machines, and (iii) if r satisfies $\sqrt{2} < r \leq \phi$, then $UB_v(r) < LB(r)$.*

Proof. Consider r such that $\sqrt{2} < r \leq \phi$, thus implying $r < 1 + 1/r$. Theorem 6 implies that no online monotone algorithm can be less than $LB(r)$-competitive, with $LB(r) = \min\{r, 1 + 1/r\} = r$. On the contrary, if verification is allowed, then the greedy algorithm is weakly monotone. Theorem 9 thus implies that its competitive ratio ρ_{gr} satisfies (see Table 1)

$$UB_v(r) \leq \rho_{gr} \leq 1 + 1/(r+1).$$

For $r > \sqrt{2}$, it holds that $r > 1 + 1/(r+1)$, thus implying $UB_v(r) < LB(r)$.

In [3] an 8-competitive algorithm Assign-R has been given. The algorithm assumes that the optimum $\mathsf{opt}(s, J)$ is known in advance and assigns a new job to the *least capable machine*, that is, the slowest machine such that the cost of the resulting assignment stays below $\Lambda \stackrel{def}{=} 2 \cdot \mathsf{opt}(s, J)$. A simple doubling technique is then used to remove this assumption at the cost of losing a factor of 4 in the approximation.

A simple observation shows that algorithm Assign-R is *not* weakly monotone. We next modify it so to obtain a weakly-monotone algorithm having a constant competitive ratio for the $\mathrm{Q}||\mathrm{C}_{\max}$ problem, i.e., for any (even non-constant) number of machines.

Algorithm Monotone-Assign-R(s, Λ):
/* $s_1 \leq s_2 \cdots \leq s_m$; */
initialize $w'_j := 0$ and $w''_j := 0$ for every machine j;
1. upon arrival of new job J_i do begin
2. let l be the slowest machine such that

$$((w''_l + J_i)/s_l \leq 2\Lambda) \wedge ((w'_l > 0) \vee (w'_{l+1} > 0));$$

3. assign J_i to machine l;
4. if $w'_l > 0$ then $w''_l := w''_l + J_i$ else $w'_l := J_i$; end.

Fig. 3. An online weakly monotone algorithm for any number of machines

Algorithm Monotone-Assign-R (see Fig. 3) receives a threshold Λ. In assigning the k^{th} job to a machine, the algorithm considers the slowest machine i for which the makespan of the resulting schedule, computed considering only the *real* jobs, does not exceed 2Λ. Then two cases are possible:

1. At least one machine faster than j has not received any load yet. Then job k is assigned to the fastest such machine and is considered a *ghost* job.
2. All machines faster than j have been assigned at least one job. In this case, job k is assigned to machine j and is considered a *real* job.

Lemma 1. *For every speed vector s and for every $\Lambda \geq 2 \cdot \mathsf{opt}(s, J)$, algorithm* Monotone-Assign-R *does not fail in assigning any newly arrived job in J. Moreover, if algorithm* Monotone-Assign-R *fails in assigning a job J_i, then $\mathsf{opt}(s, J) \geq \mathsf{opt}(s, J^i) \geq \Lambda$.*

Proof. Let J' denote the set of jobs that Monotone-Assign-R assigns to a machine which is currently empty, and $J'' \stackrel{def}{=} J \setminus J'$. Jobs in J'' are assigned according to algorithm Assign-R. Hence, if Monotone-Assign-R fails, then Assign-R fails as well. Therefore, $\mathsf{opt}(s, J^i) \geq \Lambda$ and the lemma follows.

Using a doubling technique (see e.g. [7]) one can obtain an algorithm Monotone-Assign-R which, starting from $\Lambda = 1$, doubles the value of Λ each time Monotone-Assign-R(s, Λ) fails: in this case we assign J_i, and jobs that possibly arise subsequently, by running Monotone-Assign-R with a new parameter $\Lambda' = 2\Lambda$. (We continue doubling the value of Λ until it is possible to assign J_i to some machine.) Notice that every time we double the value of Λ, we ignore the assignment performed in the previous phases (i.e., for smaller values of Λ).

Theorem 11. *Algorithm* Monotone-Assign-R *is at most* 12*-competitive.*

Proof. Let J' denote the set of jobs that Monotone-Assign-R assigns to a machine which is currently empty, and $J'' \stackrel{def}{=} J \setminus J'$. Let $\Lambda(s, J)$ denote the last value for which Monotone-Assign-R does not fail. Algorithm Monotone-Assign-R assigns jobs in J'' as algorithm Assign-R. Moreover, each machine has at most one extra job from J'. Hence, given the values w'_j and w''_j defined as in algorithm Monotone-Assign-R (see Fig. 3), we have $w'_j \leq \Lambda$ and $w''_j \leq \Lambda$. From Lemma 1 we obtain $w''_j \leq 8 \cdot \mathsf{opt}(s, J')$ and $w'_j \leq 4 \cdot \mathsf{opt}(s, J'')$. Hence, at each time step, the cost C of the solution satisfies $C \leq \max_{1 \leq j \leq m}\{w'_j + w''_j\} \leq 12 \cdot \mathsf{opt}(s, J)$.

Theorem 12. *Algorithm* Monotone-Assign-R *is weakly-monotone.*

Proof. Given the speed vector s, let $s' = (s'_i, s_{-i})$ with $s'_i < s_i$. We denote by $\Lambda(s, i)$ the value of Λ for which Monotone-Assign-R allocates job J_i. We will prove by induction on i that $\Lambda(s, i) = \Lambda(s', i)$ and that Monotone-Assign-R produces the same allocation. The base step $i = 1$ is trivial. As for the inductive step, since J_i is not allocated to machine with speed s_i, let l be the index of the machine to which J_i is allocated to. If $(w''_l + J_i)/s_l \leq \Lambda(s, i-1)$, then, by inductive hypothesis,

the same holds with respect to s', thus implying that J_i is also allocated to machine l on input s'. Clearly, in this case, $\Lambda(s', i) = \Lambda(s', i-1) = \Lambda(s, i)$. Otherwise, let $l(s)$ and $l(s')$ denote the index of the machine to which job J_i is assigned to on input s and s', respectively. In the two cases, we must increase the corresponding threshold up to a value such that $(w''_{l(s)} + J_i)/s_{l(s)} \leq \Lambda(s, i)$ and $(w''_{l(s')} + J_i)/s_{l(s')} \leq \Lambda(s', i)$. Hence, $\Lambda(s', i) = \Lambda(s, i)$ and $l(s') = l(s)$. (The latter equality follows from Step 2 in Monotone-Assign-R(s, Λ)). By inductive hypothesis, the allocation of J^{i-1} is the same, thus implying that also job J_i is allocated to the same machine.

Finally, using the payment functions for weakly monotone algorithms of [6], we can obtain the following:

Corollary 4. *The* Q$||$C$_{\max}$ *problem with verifiable machines admits an online truthful polynomial-time mechanism which is* 12*-competitive.*

Acknowledgements. We are grateful to the authors of [1] for providing us with a copy of their work.

References

1. N. Andelman, Y. Azar, and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *Annual Symposium on Theoretical Aspects of Computer Science (**STACS**)*, volume 3404 of *LNCS*, pages 69–82, 2005.
2. A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the IEEE Symposium on Foundations of Computer Science (**FOCS**)*, pages 482–491, 2001.
3. J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.
4. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *Annual Symposium on Theoretical Aspects of Computer Science (**STACS**)*, volume 2996 of *LNCS*, pages 608–619. Springer, 2004.
5. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. On designing truthful mechanisms for online scheduling. Technical report, European Project CRESCCO, http://www.ceid.upatras.gr/crescco/, 2004.
6. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. The power of verification for one-parameter agents. In *International Colloquium on Automata, Languages, and Programming (**ICALP**)*, volume 3142 of *LNCS*, 2004.
7. Y. Azar. *Online load balancing.* Springer, 1998. In Online algorithms - the state of the art, pag. 178-195.
8. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
9. N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (**STOC**)*, pages 129–140, 1999.
10. W. Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance*, pages 8–37, 1961.

On Private Computation in Incomplete Networks*

Amos Beimel

Dept. of Computer Science, Ben Gurion University,
Beer Sheva 84105, Israel

Abstract. Suppose that some parties are connected by an incomplete network of reliable and private channels. The parties cooperate to execute some protocol. However, the parties are curious – after the protocol terminates each processor tries to learn information from the communication it heard. We say that a function can be computed privately in a network if there is a protocol in which each processor learns only the information implied by its input and the output of the protocol. The question we address in this paper is what functions can be computed privately in a given incomplete network. It is known that if a network is 2-connected then every pair of parties can communicate privately. Thus, the question is interesting only for non-2-connected networks. We first characterize the functions that can be computed privately in simple networks – networks with one separating vertex and two 2-connected components. We then deal with private computations in arbitrary networks: we reduce this question to private computations of related functions on trees, and give sufficient and necessary conditions on the functions that can be computed privately on trees.

1 Introduction

The question of private computation of functions on communication networks is a fundamental question. For example, we would like to compute the output of an electronic election without revealing the votes of individuals. The general scenario we consider is that some parties are connected by an incomplete network of reliable and private channels and each party has an input. The parties cooperate to honestly execute some protocol computing a given function, but some of them are curious. That is, after the protocol terminates they collude and try to learn information from the communication they heard. A protocol is t-private if any coalition of at most t passively corrupted parties does not learn any information that is not implied by their inputs and the output of the function.

Many papers dealing with private computation, e.g., [7, 3, 8], assume that the communication network is complete, that is, there is a private and reliable communication channel between any pair of parties. The question we address in

* Partially supported by the Lynn and William Frankel Center for Computer Sciences.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 18–33, 2005.

this paper, following [4], is what functions can be computed privately in a given incomplete network. If the network is sufficiency connected then the situation is simple as proved by [3, 7, 11, 12].

Theorem 1. *If $n > 2t$ and the network is $(t+1)$-connected, every function can be computed t-privately in G.*

Bläser et al. [4] characterize the Boolean functions that can be 1-privately computed in connected networks with one separating vertex and two 2-connected components. We consider the more general question that naturally arises. Our goal is, given a communication network, characterize which functions can be computed 1-privately in this network.

Our Results. We first consider simple networks with one separating vertex and two 2-connected components. We give an exact characterization of the functions that can be computed 1-privately in such a network. This result generalizes a result of Bläser et al. [4] who characterize the *Boolean* functions that can be computed 1-privately in such a network. While Boolean functions that can be computed privately in such networks are very simple structure ("if then else" functions), the non-Boolean functions that can be computed privately in such networks have a richer structure. Our proof is somewhat simpler than the proof of [4], and has two stages: We first reduce the question of private computation in such a network to a question of private computation of a related function with two variables in a simpler model, and then characterize the functions that can be computed in the simpler model.

We next consider 1-private computations in arbitrary networks. We reduce the private computation of a function in an arbitrary network to private computation of a related function in a tree. The idea of this reduction is that we can replace each 2-connected component in the network by a single vertex holding the inputs of the component. We then give sufficient and necessary conditions on the functions that can be computed privately on trees. We do not know the exact characterization of the functions that can be computed privately on trees.

t-privacy. In this work we focus on 1-privacy. Our results in Section 3 generalize to networks with one separating set of size $t-1$ and two t-connected components. However, our results for arbitrary networks do not generalize to t-privacy as the component structure of such networks can be complicated.

Historical Notes. There are a few models of secure computation. One distinction is whether the "bad" parties have unlimited power or they are polynomial-time randomized machines. The other distinction is whether the "bad" parties are honest-but-curious, or they are malicious, that is, they deviate from their protocol to gain more information. In this work we consider honest-but-curious parties with unlimited power. We review some previous results concerning this model. Chaum, Crépeau, and Damgård [7] and Ben-or, Goldwasser, and Wigderson [3] proved that in a complete network with n parties, if $n > 2t$ then every function can be computed t-privately. Kushilevitz [20] characterizes the functions that

can be computed privately in a network with two parties. Chor and Kushilevitz [8] characterize the Boolean functions that can be computed t-privately in complete networks when $n \leq 2t$. All these works, as well as our work, assume that the network is synchronous.

We next consider private computation in incomplete networks. Dolev, Dwork, Waarts, and Yung [12] have proved that if there are at most t honest-but-curious parties, then every pair of parties can communicate privately if and only if the network is $(t+1)$-connected. Bläser, Jakoby, Liśkiewicz, and Manthey [4], in a work that inspired the current work, characterize the *Boolean* functions that can be computed 1-privately in a network with one separating vertex and two 2-connected components. They also considered the randomness required for private protocols in incomplete networks. Jakoby, Liskiewicz, and Reischuk [18] considered tradeoff between randomness and connectivity in private computation. Finally, Bläser et al. [5] consider protocols that reveal minimum information for functions that cannot be computed privately in a given incomplete network.

The connectivity requirements for several distributed tasks in several models has been studied in many papers; for example Byzantine agreement [11, 14], approximate Byzantine agreement [13, 27], reliable message transmission [11, 12], and reliable and private message transmission [23, 12, 24, 25, 26]. Simple impossibility results and references can be found in [14, 21]. Connectivity requirements in partially authenticated networks has been considered in [1, 2]. Secure communication and secure computation in multi-recipient (multi-cast) models have been studied in [17, 16, 15, 9]. Secure computation in directed networks has been studied in [10]. Secure communication against general adversarial structures has been studied in [19].

Organization. In Section 2, we describe our model and present some background on connectivity. In Section 3, we characterize the functions that can be computed 1-privately in networks with two 2-connected components. In Section 4, we reduce private computation of functions in arbitrary networks to private computation of related function on trees, and, in Section 5, we give sufficient and necessary conditions on the functions that can be computed privately on trees.

2 Preliminaries

The Model. The communication network is modeled by an undirected graph $G = \langle V, E \rangle$, where (1) The vertices $V = \{v_1, v_2, \ldots, v_n\}$ are the parties in the network. We denote their number by n (i.e., $|V| = n$); in the sequence we refer to parties as vertices. (2) The edges E describe the communication channels. That is, there is an edge $\langle u, v \rangle$ in E if and only if there is a communication channel between u and v. We assume that these communication channels are reliable and private: an adversary that does not control u or v (but might control all other vertices in the network) cannot read, change, delete, or insert messages sent on the edge $\langle u, v \rangle$.

Protocols. We consider an n-party protocol for computing a given function. Briefly, in the beginning of the protocol, each vertex v_i has a private *input* a_i and a private *random input* r_i, where r_i is distributed uniformly in some finite domain (the random inputs $(r_1, \ldots, r_n)$ are independent). A protocol Π computes its output in a sequence of rounds. For a round j, let $i \leftarrow (j \bmod n) + 1$. In round j, only Vertex v_i is active[1] and sends a message $m_{j,k}$ (i.e., a string) to v_k for each of its neighbors; this message will become an available input to v_k in the next round. If v_k is not a neighbor of v_k then $m_{j,k}$ is the empty string. The message $m_{j,k}$ is a function of the round number j, the receiver k, the sender's input a_i, the sender's random input r_i, and the previous messages v_i got, i.e., $\langle m_{j',i} \rangle_{1 \leq j' < j}$. A computation of the protocol ends in a round in which each vertex computes an *output*.

Transcripts, Views, and Outputs. Let $S \subseteq \{v_1, \ldots, v_n\}$. Given an execution of a protocol Π on inputs $(a_1, \ldots, a_n)$ and random inputs $(r_1, \ldots, r_n)$, we define: The *transcript* of S of the execution is the sequence of messages that vertices in S got during the execution; it is denoted by $\text{TRANS}_S(a_1, \ldots, a_n, r_1, \ldots, r_n)$. The *view* of S is the triplet $\langle a_i \rangle_{v_i \in S}$, $\langle r_i \rangle_{v_i \in S}$, and $\text{TRANS}_S(a_1, \ldots, a_n, r_1, \ldots, r_n)$; it is denoted by $\text{VIEW}_S(a_1, \ldots, a_n, r_1, \ldots, r_n)$. We consider the random variables $\text{TRANS}_S(a_1, \ldots, a_n, \langle r_i \rangle_{v_i \in S})$ obtained by randomly selecting $\langle r_i \rangle_{v_i \notin S}$ and outputting $\text{TRANS}_S(a_1, \ldots, a_n, r_1, \ldots, r_n)$. We also consider the similarly defined random variables for $\text{VIEW}_S(a_1, \ldots, a_n, \langle r_i \rangle_{v_i \in S})$.

In the model we consider, the n-party honest-but-curious model, each party is curious, that is, coalitions of parties may try to deduce as much information possible from their own view of an execution about the private inputs of the other parties. However, each party is honest, that is, it scrupulously follows the instructions of the protocol. In such conditions, it is easy to enforce the correctness condition (for securely computing a function f), but not necessarily the privacy conditions.

In the following definition we consider functions $f : A_1 \times \ldots \times A_n \rightarrow O$, where $A_1, \ldots, A_n$ and O are some finite sets, and the ith input of f is the input of v_i. The privacy requirement we consider is unconditional, that is, even a curious adversary with unlimited power does not gain information. Furthermore, we consider perfect security, that is, we require no error in the correctness, and exactly the same distributions in the privacy requirement. Our results remain the same if we only require statistical security. In the following definition we define privacy against an adversarial structure $\mathcal{S} \subseteq 2^V$, that is $\mathcal{S}$ is a collection of subsets of the vertices. We require that if parties in a set in $\mathcal{S}$ collude then, from their view, they do not gain information that is not implied by their inputs and the output of the function. In this work we mainly focus on 1-privacy, that is we want to protect the privacy against each individual. We define the more general case of $\mathcal{S}$-privacy as it is used as a tool to characterize 1-privacy.

[1] By adding extra rounds, this assumption is without loss of generality.

Definition 1 (Private Computation). *Let $G = \langle V, E\rangle$ be network with n vertices, $A_1, \ldots, A_n$, and O be finite sets, $f : A_1 \times \ldots \times A_n \to O$ be a function, and $\mathcal{S} \subseteq 2^V$ be an adversarial structure. A protocol Π* $\mathcal{S}$-privately computes *f, if the following conditions hold:*

Correctness. *For every $a_1, \ldots, a_n$ and every $r_1, \ldots, r_n$, the output of each v_i with $\text{VIEW}_{\{v_i\}}(a_1, \ldots, a_n, r_1, \ldots, r_n)$ is $f(a_1, \ldots, a_n)$.*

Privacy. *For every $S \in \mathcal{S}$, for every $\langle a_1, \ldots, a_n\rangle \in A_1 \times \ldots \times A_n$ and every $\langle a'_1, \ldots, a'_n\rangle \in A_1 \times \ldots \times A_n$ such that $a_i = a'_i$ for every $v_i \in S$, and every $\langle r_i\rangle_{v_i \in S}$, if $f(a_1, \ldots, a_n) = f(a'_1, \ldots, a'_n)$ then the random variables $\text{VIEW}_S((a_1, \ldots, a_n, \langle r_i\rangle_{v_i \in S}))$ and $\text{VIEW}_S((a'_1, \ldots, a'_n, \langle r_i\rangle_{v_i \in S}))$ are equally distributed.*

A function f can be computed t-privately in G if there is a protocol Π that $\mathcal{S}$-privately computes f in G, where $\mathcal{S} = \{S \subseteq V : |S| \leq t\}$.

Since each vertex learns the output of f (and knows its input), we require that the privacy is protected only when $f(a_1, \ldots, a_n) = f(a'_1, \ldots, a'_n)$. We assume that all parties in the system know the topology of the graph G. Furthermore, we assume that the system is synchronous and all the parties in the system know in which round the protocol starts.

In the sequence, we will use the following proposition of [8], which holds for every 2-party protocol (i.e., even without the correctness and privacy requirement). Informally, this proposition says that if changing the inputs of both parties yields the same transcript, then changing the input of only one party also yields the same transcript.

Proposition 1 ([8]). *Consider a two-party protocol. Let a_1, a_2, a'_1, a'_2 be inputs, r_1, r'_1, r_2, r'_2 be random inputs, and h be a transcript such that*

$$\text{TRANS}_{\{1,2\}}(a_1, a_2, r_1, r_2) = \text{TRANS}_{\{1,2\}}(a'_1, a'_2, r'_1, r'_2) \;=\; h.$$

Then, $\text{TRANS}_{\{1,2\}}(a_1, a'_2, r_1, r'_2) = \text{TRANS}_{\{1,2\}}(a'_1, a_2, r'_1, r_2) = h.$

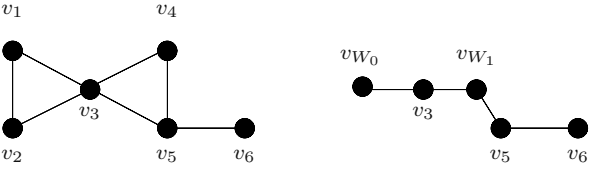
Connectivity. The reliability of a network is closely related to its connectivity. In this section, we review the relevant concepts related to connectivity. For more details, the reader can consult, e.g., [6].

We consider *vertex* connectivity of *undirected* graphs. A graph $G = \langle V, E\rangle$ is connected if for every two vertices u, v there is a path connecting them in G. In this paper, we only consider connected graphs. A vertex $z \in V$ is called a *separating vertex* (or a cut-vertex) if for some $u, v \in V \setminus \{z\}$ every path between u and v passes through z. For a connected graph, a vertex z is a separating vertex if and only if removing z from G results in an unconnected graph. A connected graph is 2-connected if it contains at least 3 vertices and it does not contain a separating vertex. By a result of Menger [22], a graph G with at least 3 vertices is 2-connected if and only if for every vertices $u, v \in V$ either $\langle u, v\rangle \in E$ or there exist two vertex-disjoint paths between u and v in G. An edge e is a *bridge* if for some $u, v \in V$, every path between u and v passes through e.

A subgraph B of G is a *component* if it is a maximal 2-connected induced subgraph of G. We next define the component graph of a connected graph, which replaces every component in G by a single vertex.[2]

Definition 2 (Component Graph). *Given a connected graph $G = \langle V, E \rangle$, we define its component graph $T_G = \langle V', E' \rangle$ as follows: The vertices in V' are the components of G, the leaves in G, and the separating vertices in G. There is an edge in E' between every separating vertex and every component containing it, between a leaf and its neighboring separating vertex, and between two separating vertices connected by a bridge. For every component W in G, we denote the corresponding vertex in T_G by v_W.*

For example, a graph G and its component graph are described Fig. 1. In G there are two components $W_0 = \{v_1, v_2, v_3\}$ and $W_1 = \{v_3, v_4, v_5\}$, two separating vertices v_3 and v_5, and one leaf v_6. Thus, the component graph of G has 5 vertices.



A graph G. The component graph of G.

Fig. 1. A graph and its component graph

By Menger's theorem, every cycle in G is contained in exactly one component. This fact implies the following observation.

Observation 2. *If the graph G is connected, then the graph T_G is a tree.*

3 Incomplete Networks with Two 2-Connected Components

In this section we characterize the functions that can be computed 1-privately in connected networks that contain one separating vertex and two 2-connected components. As an intermediate step, we consider a model we call the two-party and eavesdropper model. Using this intermediate model, we characterize the functions that can be computed privately in connected networks that contain two 2-connected components. That is, we prove that a function can be computed privately in connected networks that contain two 2-connected components if and

[2] The component graph we define is similar to the block-cutvertex graph as defined in [6].

only if a related function can be computed in the two-party and eavesdropper model. Roughly speaking, the two parties correspond to the 2-connected components and the eavesdropper is the separating vertex. To complete the characterization, we characterize the functions that can be computed privately in the the two-party and eavesdropper model.

The Two-Party and Eavesdropper Model. Consider the following two-party and eavesdropper model of private computation. Alice has a secret input a taken from some finite domain A and Bob has a secret input b taken from some finite domain B; they wish to compute a function $f : A \times B \to O$ such that the eavesdropper Eve, which hears the communication that they exchange, can compute $f(a, b)$; however Eve should not learn any information on a and b that is not implied by $f(a, b)$. Formally, we consider the network with 3 vertices $\{v_1, v_2, v_3\}$ (where v_1 and v_3 are Alice and Bob respectively and v_2 is Eve) and two edges $\langle v_1, v_2\rangle$ and $\langle v_2, v_3\rangle$, consider functions that do not depend on v_2's input, and consider the adversarial structure $\mathcal{S} = \{\{v_2\}\}$.

3.1 Reduction to the Two-Party and Eavesdropper Model

We first use the two-party and eavesdropper model to characterize the functions that can be 1-privately computed in connected networks with two 2-connected components. We consider a network G_{n_0,n_1}, with $n_0 + n_1 + 1$ vertices, which is composed of two 2-connected components. The first component is denoted by W_0 and has $n_0 + 1$ vertices; the second component is denoted by W_1 and has $n_1 + 1$ vertices. The two components share exactly one vertex denoted z. In this paper we assume that $n_0, n_1 \geq 2$, that is, each connected component contains at least 3 vertices. (The cases where $n_0 = 1$ or $n_1 = 1$ is characterized in the full version of this paper). Such a graph is illustrated in Fig. 2. Given an (n_0+n_1+1)-argument function $f : A_1 \times \ldots \times A_{n_0} \times B_1 \times \ldots \times B_{n_1} \times C \to O$, define, for every $c \in C$, a 2-argument function $f_c : (A_1 \times \ldots \times A_{n_0}) \times (B_1 \times \ldots \times B_{n_1}) \to O$, where for every $\boldsymbol{a} \in A_1 \times \ldots \times A_{n_0}$ and every $\boldsymbol{b} \in B_1 \times \ldots \times B_{n_1}$,

$$f_c(\boldsymbol{a}, \boldsymbol{b}) \stackrel{\text{def}}{=} f(\boldsymbol{a}, \boldsymbol{b}, c). \tag{1}$$

Lemma 1. *Let $f : A_1 \times \ldots \times A_{n_0} \times B_1 \times \ldots \times B_{n_1} \times C \to O$ be a function, where $n_0, n_1 \geq 2$. The function f can be computed 1-privately in G_{n_0,n_1} if and only if for every $c \in C$ the function f_c can be computed privately in the two-party and eavesdropper model, where Alice's input is $\boldsymbol{a}$ and Bob's input is $\boldsymbol{b}$.*

Proof. First, assume that there is a protocol privately computing f in G_{n_0,n_1}. For every $c \in C$, we construct a private protocol for f_c in the two-party and eavesdropper model. Alice, holding $\boldsymbol{a} \in A_1 \times \ldots \times A_{n_0}$ simulates the $n_0 + 1$ vertices in the component W_0 (including z), and Bob, holding $\boldsymbol{b} \in B_1 \times \ldots \times B_{n_1}$ simulates the n_1 vertices in component W_1 excluding z. At the end of the protocol, Alice sends the output to Bob, thus the eavesdropper knows the output. The eavesdropper knows that the input of z is c, as c is fixed, and hears

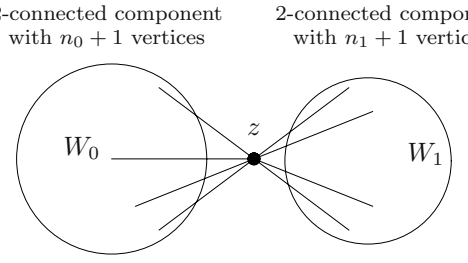


Fig. 2. The Graph G_{n_0,n_1}

the messages exchanged between z and the vertices in W_1. Thus, the information the eavesdropper learns is at most the information that z learns in the protocol for f, and the 1-privacy of that protocol implies the privacy of the protocol for f_c.

Now assume that, for every $c \in C$, the function f_c can be privately computed in the two-party and eavesdropper model, where Alice's input is $\boldsymbol{a}$ and Bob's input is $\boldsymbol{b}$. By Corollary 1 (appearing in Section 3.2), we can assume that this protocol is deterministic. We construct a (randomized) protocol for f. W.l.o.g., assume that the protocol Π_c for every f_c proceeds in rounds, where in odd rounds Alice sends a one bit message to Bob, and in even rounds Bob sends a one bit message to Alice. Let Π_c^i be the ith message sent in the protocol. Thus, in odd rounds (respectively, even rounds) the bit Π_c^i depends on c, $\boldsymbol{a}$ (respectively, $\boldsymbol{b}$), and the previous messages. The protocol for f will have a virtual round for each round of the protocol for f_c. In each virtual round, Vertex z picks a random bit r_i, and the parties in $W_{i \bmod 2}$ (including z) compute the function $\Pi_c^i \oplus r_i$ using a 1-private protocol. Such 1-private protocol exists by Theorem 1, since $1 + n_{i \bmod 2} > 2$ and each component is 2-connected.

We next argue that this protocol is 1-private. As the vertices use 1-private protocols to compute each value $\Pi_c^i \oplus r_i$ and r_i is chosen at random by z, each vertex, except for z, does not learn any information during the protocol. Vertex z knows the random bits, thus, it knows the communication exchanged in the protocol for f_c. However, the information it gets is exactly the information the eavesdropper gets in the protocol for f_c, thus z gains no information. □

3.2 The Two-Party and Eavesdropper Model

The functions that can be computed privately in the two-party and eavesdropper model are a subset of the functions that can be computed in the two-party model (without the eavesdropper) as characterized by Kushilevitz [20]. We first introduce some notation from [20]. We represent a function $f : A \times B \to O$ by a matrix M_f whose rows are labeled by the elements of A, columns are labeled by the elements of B, and $M_f(a, b) = f(a, b)$.

Definition 3 ([20]). *Let M be a matrix whose rows are labeled by the elements of A and columns are labeled by the elements of B. The relation $\sim_C$ on B is defined as follows: $b, b' \in B$ satisfy $b \sim_C b'$ if there exists some $a \in A$ such that $M(a,b) = M(a,b')$. The equivalence relation $\equiv_C$ on B is defined as the transitive closure of the relation $\sim_C$. That is, $b \equiv_C b'$, for $b, b' \in B$, if there are $b_1, \ldots, b_\ell$ such that $b \sim_C b_1 \sim_C b_2 \sim_C \ldots \sim_C b_\ell \sim_C b'$. Similarly, the relations $\sim_R$ and $\equiv_R$ are defined on A. That is, $a, a' \in A$ satisfy $a \sim_R a'$ if there exists some $b \in B$ such that $M(a,b) = M(a',b)$, and the relation $\equiv_R$ on A is defined as the transitive closure of the relation $\sim_R$. If $b \equiv_C b'$, then we say that columns b and b' of M are equivalent, and, similarly, if $a \equiv_R a'$, then we say that rows a and a' of M are equivalent.*

Definition 4 (Forbidden Matrix [20]). *A matrix M is a forbidden matrix if the following three conditions hold: (1) the matrix is not constant, (2) all the rows of M are equivalent according to $\equiv_R$, and (3) all the columns of M are equivalent according to $\equiv_C$.*

Kushilevitz [20] proved that a function f can be privately computed in the two-party model (without the eavesdropper) if and only if the matrix M_f does not contain a forbidden matrix. We adapt this result to the two-party and eavesdropper model, where there is an additional requirement.

Lemma 2. *A function $f : A \times B \to O$ can be computed privately in the two-party and eavesdropper model if and only if*

1. *The inputs corresponding to any output value form a rectangle. That is, for every $a_0, a_1 \in A$, every $b_0, b_1 \in B$, and every $o \in O$ if $f(a_0, b_0) = f(a_1, b_1) = o$ then $f(a_0, b_1) = f(a_1, b_0) = o$.*
2. *The matrix M_f does not contain a forbidden matrix.*

Proof. First, assume that f satisfies Conditions (1) and (2). We construct a deterministic private protocol computing f. The protocol is identical to the protocol of [20], with a small change in the proof of privacy. In each step of the protocol, Alice, holding an input a, and Bob, holding an input b, maintain a rectangle $A_0 \times B_0 \subseteq A \times B$, known also to Eve, such that $\langle a, b \rangle \in A_0 \times B_0$. In the beginning, $A_0 \leftarrow A$ and $B_0 \leftarrow B$. At the end of the protocol, $A_0 \times B_0$ is constant, so Eve can deduce the value of f. In each step, consider the matrix M which is the matrix M_f restricted to $A_0 \times B_0$. The matrix M becomes smaller in each step, and the equivalence relations $\equiv_R$ and $\equiv_C$ change accordingly. By Condition (2), the matrix M is not forbidden. If M is constant, Eve deduces that this constant is the output and the protocol ends. If not all the rows of M are equivalent according to $\equiv_R$, Alice sends to Bob the equivalence class of a in M, and both parties set A_0 as this equivalence class. Otherwise, not all the columns of M are equivalent according to $\equiv_C$, Bob sends to Alice the equivalence class of b in M, and both parties set B_0 as this equivalence class. As M_f does not contain a forbidden matrix, the protocol must reach a constant rectangle. Since, in each stage of the protocol, $\langle a, b \rangle \in A_0 \times B_0$, this protocol

is correct. We next argue that Eve does not learn information on $\langle a, b\rangle$ that is not implied by $f(a, b)$. This follows from the fact that if $f(a, b) = f(a', b')$ then, by Condition (1), $f(a, b) = f(a, b') = f(a', b) = f(a', b')$. Thus, in each stage of the protocol, $a \equiv_R a'$ and $b \equiv_R b'$ in M, and the same communication string is exchanged on $\langle a, b\rangle$ and $\langle a', b'\rangle$, thus Eve does not gain extra information.

We next assume that Conditions (1) and (2) are necessary. It can be shown that if f can be computed privately in the two-party and eavesdropper model, then it can be computed privately in the regular two-party model. Thus, by [20], the matrix M_f does not contain a forbidden sub-matrix.

Suppose that the set of inputs corresponding to some output value is not a rectangle, that is, there are $a_0, a_1 \in A$, and $b_0, b_1 \in B$ such that $f(a_0, b_0) = f(a_1, b_1) = o$ while $f(a_0, b_1) \neq o$. Since Eve does not learn any information on the inputs, the probability distribution on the transcripts that are possible on $\langle a_0, b_0\rangle$ is equivalent to the probability distribution on the transcripts that are possible on $\langle a_1, b_1\rangle$. By Proposition 1, this distribution should be the same on the inputs $\langle a_0, b_1\rangle$, contradicting the requirement that Eve can compute f from the communication. □

Notice that in the proof of Lemma 2 we construct a deterministic protocol.

Corollary 1. *A function $f : A \times B \to O$ can be computed privately in the two-party and eavesdropper model if and only if it can be computed privately in the two-party and eavesdropper model by a deterministic protocol.*

We next describe two examples.

$\mathbf{f_1}$	b_0	b_1	b_2
a_0	0	0	3
a_1	2	1	1
a_2	2	4	3

$\mathbf{f_2}$	b_0	b_1	b_2
a_0	0	0	3
a_1	2	1	1
a_2	2	4	4

In both examples, Condition (1) holds. For example, in both examples the rectangle corresponding to the output value 2 is $\{a_1, a_2\} \times \{b_0\}$. In the first example, however, the matrix is forbidden and the function f_1 cannot be computed privately. In the second example, we changed the bottom-left entry from 3 to 4; now the matrix does not contain a forbidden sub-matrix, and the function f_2 can be computed privately. The partition induced by the protocol is detailed in the matrix.

The next lemma, which is implicit in [4], states that the characterization for Boolean functions is much simpler.

Lemma 3. *A Boolean function $f : A \times B \to \{0, 1\}$ can be computed privately in the two-party and eavesdropper model iff it depends only on one of its inputs, that is, if there exists a function f' such that at least one of the following conditions hold: (1) $f(a, b) = f'(a)$ for all $a \in A$ and $b \in B$, or (2) $f(a, b) = f'(b)$ for all $a \in A$ and $b \in B$.*

Proof. If a function f (Boolean or non-Boolean) depends only on one of its inputs then it satisfies Conditions (1) and (2) of Lemma 2, thus can be computed privately in the two-party and eavesdropper model.

For the other direction, assume that a Boolean function f can be computed privately in the two-party and eavesdropper model, thus satisfies Condition (1). Assume towards contradiction that f depends on its two input, thus: (1) As f depends on its first input, there exist $b \in B$ and $a', a'' \in B$ such that $f(a', b) = 0$ and $f(a'', b) = 1$, and (2) As f depends on its second input, there exist $a \in A$ and $b', b'' \in B$ such that $f(a, b') = 0$ and $f(a, b'') = 1$. By Condition (1), on one hand, $0 = f(a, b') = f(a', b) = f(a, b)$, and on the other hand $1 = f(a, b'') = f(a'', b) = f(a, b)$, a contradiction. □

Combing Lemma 1 and Lemma 2, we get a combinatorial characterization of the functions that can be computed 1-privately in networks with two components.

Theorem 3. *Let $f : A_1 \times \ldots \times A_{n_0} \times B_1 \times \ldots \times B_{n_1} \times C \to O$ be a function, where $n_0, n_1 \geq 2$. The function f can be computed 1-privately in G_{n_0,n_1} iff for every $c \in C$:*

1. *The inputs of f_c (as defined in (1)) corresponding to any output value form a rectangle. That is, for every $\boldsymbol{a_0}, \boldsymbol{a_1}$ and $\boldsymbol{b_0}, \boldsymbol{b_1}$, if $f_c(\boldsymbol{a_0}, \boldsymbol{b_0}) = f_c(\boldsymbol{a_1}, \boldsymbol{b_1}) = o$ then $f_c(\boldsymbol{a_0}, \boldsymbol{b_1}) = f_c(\boldsymbol{a_1}, \boldsymbol{b_0}) = o$.*
2. *The matrix M_{f_c} does not contain a forbidden matrix.*

4 Networks with Many Connected Components

In this section we consider private computation of functions in arbitrary connected networks. As in the previous section, the characterization of the functions that can be computed privately has two stages. We first reduce the problem of private computation in the network to private computation of a related function in the component graph of G, which is a tree. In Section 5, we give necessary and sufficient conditions for computing a function privately on trees. However, we do not give an exact characterization of these functions.

4.1 Reduction to Private Computation in Trees

In this section we reduce private computation in an arbitrary connected network to private computation of a related function in a tree, namely, the component graph of the network. Formally, let G be a graph with n vertices and T_G be the component graph of G with n' vertices (as defined in Definition 2). We say that a vertex in G is curious if it is either a separating vertex in G or a leaf in G. Notice that curious vertices in G are also vertices in T_G. Given a function $f : A_1 \times \ldots \times A_n \to O$ we define an n'-argument function f', where the input of a curious vertex is the same as before and the input of a vertex v_W, for a component W in G, is the inputs of the non-separating vertices in W.

In the component graph we replaced every component W in G by one vertex v_W in T_G holding the inputs of the non-separating vertices in the component. The idea of the reduction is that in G we can compute by a private protocol the messages sent by v_W in the tree. Hence, we do not need any privacy requirements for such v_W.

Lemma 4. *Let $\mathcal{S} = \{\{v\} : v$ is a curious vertex in $G\}$. A function f can be computed 1-privately in G iff f' can be computed $\mathcal{S}$-privately in T_G.*

Proof. First, assume that there is a protocol Π privately computing f in G. We construct an $\mathcal{S}$-private protocol Π' computing f' in T_G. The protocol Π' simulates the protocol Π: (1) Every curious vertex in G, which is a vertex in T_G having the same input, sends and receives the same messages in both protocols. (2) Every vertex v_W simulates all the non-separating vertices in W. (3) Every message sent between two separating vertices in the same component W in G, is sent via v_W in Π'. Thus, every curious vertex has the same view in Π' as it had in Π. Since Protocol Π is 1-private and in Π' we require privacy only for the curious vertices, Protocol Π' is $\mathcal{S}$-private.

Now, assume that there is an $\mathcal{S}$-private protocol Π' computing f' in T_G. We construct a 1-private protocol Π computing f in G. In this protocol, every curious vertex will effectively have the same information as in Π', and the messages received by a vertex v_W will be secret-shared by all vertices in W. In Π', every vertex v_W has a random input r_W distributed uniformly in some finite set R. In the beginning of Protocol Π, each vertex $w \in W$ chooses a random input $r_{w,0}$ distributed uniformly in R, and the parties define $r_W = \bigoplus_{w \in W} r_{w,0}$. Protocol Π' has rounds and in each round only one vertex sends messages. W.l.o.g., assume that every message in Π' is one bit. Protocol Π will have a virtual round for every round of Π'. There are three cases to consider:

If the sender of a message m in Π' is u and the receiver is v, where u and v are curious vertices, then u sends the message m to v in Π.

If the sender of a message m in Π' is u, where u is a separating vertex in G, and the receiver is v_W, where W is a connected component in G, then each vertex w in W chooses at random, with uniform distribution, a bit r_w and the vertices in W compute the function $m \oplus \bigoplus_{w \in W} r_w$ using a 1-private protocol. By Theorem 1 such protocol exists since each connected component has size at least 3. On one hand, the vertices in W collectively know the message m. On the other hand, each vertex gains no information from this virtual round.

The last case is when the sender of a message m in Π' is v_W, where W is a connected component in G, and the receiver is v, where v is a separating vertex in G. In this case, the message m in Π' is function of the inputs of the non-separating vertices in W, the random input r_W, and the messages v_W got in previous rounds. In Protocol Π, the vertices in W know the inputs of the non-separating vertices in W, and collectively know the random input r_W and the messages v_W got in previous rounds. Thus, m is a function of inputs known to vertices in W. In Protocol Π, the receiver v chooses a random bit r_v with uniform distribution, and the vertices in W compute the function $m \oplus r_v$ using a 1-private protocol. On one hand, vertex v learns the message m, but

no additional information. On the other hand, each vertex in $W \setminus \{v\}$ gains no information from this virtual round.

We next argue that this protocol is 1-private. First we argue that every non-curious vertex in G learns no information in this protocol. The messages such a vertex gets during the execution of Protocol Π are messages in 1-private protocols computing a function masked by r_v for a separating vertex v in the connected component. Thus, each non-curious vertex does not learn any information during the protocol for f. Every curious vertex learns only the messages it got in Π', and, since Π' is $\mathcal{S}$-private, the curious vertex gains no information that is not implied by its input and the output of the function. □

5 Private Computation on Trees

By Lemma 4, to characterize which functions can be computed 1-privately on G, we need to characterize which functions can be computed $\mathcal{S}$-privately in T_G. We do not have an exact characterization of these functions. We only give necessary and sufficient conditions for this task. In the sequence, we say that a vertex v is curious if $\{v\} \in \mathcal{S}$.

5.1 Sufficient Condition

In this section we give a sufficient condition for computing a function $\mathcal{S}$-privately in a tree. Using Lemma 4, the results of this section give a sufficient condition for computing a function 1-privately in arbitrary networks. The sufficient condition is a simple generalization of the condition of [20]. We next introduce some notation generalizing Definitions 3 and 4 (taken from [20]). We represent a function $f : A_1 \times \ldots A_n \to O$ by an n-dimensional array M_f whose ith-dimension is labeled by the elements of A_i, and $M_f(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$.

Definition 5 (Forbidden Array). *Let M be an n-dimensional array whose ith-dimension is labeled by the elements of A_i, and $\mathcal{S}$ be a the collection of curious vertices (where $|S| = 1$ for every $S \in \mathcal{S}$). The relation $\sim_i$ on A_i is defined as follows: $a, b \in A_i$ satisfy $a \sim_i b$ if there exist some $\boldsymbol{a}, \boldsymbol{b} \in A_1 \times \cdots \times A_n$ and an index $j \neq i$ such that (1) $\{v_j\} \in \mathcal{S}$, (2) $a_i = a$, (3) $b_i = b$, (4) $a_j = b_j$, and (5) $M(\boldsymbol{a}) = M(\boldsymbol{b})$. The equivalence relation $\equiv_i$ on A_i is defined as the transitive closure of the relation $\sim_i$.*

An array M is a forbidden array iff (1) the array is not constant, (2) for all i, all the elements of A_i are equivalent in M according to $\equiv_i$.

Lemma 5. *Let f be a function. If the array M_f does not contain a forbidden array, then f can be computed $\mathcal{S}$-privately on any tree with n vertices.*

Proof. The protocol is a simple generalization of the protocol of [20]. In each step of the protocol, the parties, maintain a cube $R_1 \times \ldots \times R_n \subseteq A_1 \times \ldots \times A_n$, such that $\langle a_1, \ldots, a_n \rangle \in R_1 \times \ldots \times R_n$. In the beginning, $R_i \leftarrow A_i$ for $i \in \{1, \ldots, n\}$.

At the end of the protocol, $R_1 \times \ldots \times R_n$ is constant, so each party can deduce the value of f. In each step, consider the array M which is the array M_f restricted to $R_1 \times \ldots \times R_n$. As M_f does not contain a forbidden array, the array M is not forbidden. If M is constant, then all the vertices know that this constant is the output, and the protocol ends. Otherwise, for some $i \in \{1, \ldots, n\}$, not all the elements of A_i are equivalent in M according to $\equiv_i$. Vertex v_i sends to its neighbors the equivalence class of a_i in M, and this information is propagated to all vertices in the tree. Thereafter, all parties set R_i as this equivalence class. Since, in each stage of the protocol, $\langle a_1, \ldots, a_n \rangle \in R_1 \times \ldots \times R_n$, this protocol is correct. We next argue that each curious vertex v_j does not learn information on $\langle a_1, \ldots, a_n \rangle$ that is not implied by a_j and $f(a_1, \ldots, a_n)$. This follows from the fact that if v_j is curious and $f(\boldsymbol{a}) = f(\boldsymbol{b})$ where $a_j = b_j$, then in each stage of the protocol $a_i \equiv_i b_i$ in M for every i, and the same communication string is exchanged on $\boldsymbol{a}$ and $\boldsymbol{b}$, thus v_j does not gain extra information. □

In the protocol described in the proof of Lemma 5, each message is propagated to all the vertices in the tree. This was possible since the sufficient condition has strong requirements, and this explains why the sufficient condition is not necessary.

5.2 Necessary Condition

In a tree, every vertex that is not a leaf is a separating vertex. Informally, this means that such vertex can learn all the information sent from one side of a tree to the other side. Formulating this intuition is simple: We show that if a function can be computed in a tree then a related function can be computed in the two-party and eavesdropper model, where Alice's input is the inputs of one side of the tree, Bob's input is the inputs of other side of the tree, and Eve is the separating vertex. This is formulated in the next lemma, whose proof is similar to the proof of sufficiency in Lemma 1.

Lemma 6. *Let $T = \langle V, E \rangle$ be a tree and $\mathcal{S} \subseteq 2^V$, where $|S| = 1$ for every $S \in \mathcal{S}$. Let v_i be a curious vertex in T which is not a leaf. W.l.o.g. assume that for every j, k such that $j < i < k$, vertex v_i separates vertex v_j and vertex v_k. Furthermore, let f be a function that can be computed $\mathcal{S}$-privately in T. For every $c \in A_i$, define f_c as $f_c(\langle a_1, \ldots, a_{i-1} \rangle, \langle a_{i+1}, \ldots, a_n \rangle) = f(a_1, \ldots, a_{i-1}, c, a_{i+1}, \ldots, a_n)$. Then, for every every $c \in A_i$, the function f_c can be computed privately in the two-party and eavesdropper model.*

Using Lemma 2 we can deduce the following necessary condition. Roughly speaking, the condition is that the inputs corresponding to each output value are a union of certain n-dimensional cubes. For the lemma, we need the following notation: Let $f : A_1 \times \cdots \times A_n \to O$ be a function, $I \subseteq \{1, \ldots, n\}$ be a set, and $\boldsymbol{c} = \langle c_i \rangle_{i \in I}$ be a vector where $c_i \in A_i$ for every $i \in I$. Denote $f_{I,\boldsymbol{c}} : \Pi_{i \notin I} A_i \to O$, the restriction of f to $\{1, \ldots, n\} \setminus I$, as follows $f_{I,\boldsymbol{c}}(\langle a_i \rangle_{i \notin I}) = f(\langle b \rangle_{i \in \{1,\ldots,n\}})$ where $b_i = c_i$ if $i \in I$ and $b_i = a_i$ otherwise. The proof of the following lemma is omitted for lack of space.

Lemma 7. *Let $T = \langle V, E\rangle$ be a tree and $\mathcal{S} \subseteq 2^V$, where $|S| = 1$ for every $S \in \mathcal{S}$ and $\langle u, v\rangle \notin E$ for every two non-curious vertices u, v.*[3] *Assume that a function $f : A_1 \times \cdots \times A_n \to O$ can be computed $\mathcal{S}$-privately in T, and define $I \stackrel{def}{=} \{i : v_i \text{ is a curious vertex}\}$, and $n' = n - |I|$. Then, for every $\boldsymbol{c} \in \Pi_{i\in I} A_i$ and every output value $o \in O$, the inputs of $f_{I,\boldsymbol{c}}$ corresponding to o are an n'-dimensional cube, that is, there exist sets $\langle R_i\rangle_{i\notin I}$ such that $R_i \subseteq A_i$ and $f_{I,\boldsymbol{c}}(\boldsymbol{a}) = o$ if and only if $a_i \in R_i$ for every $i \notin I$.*

Acknowledgement. We would like to thank Enav Weinreb for valuable comments that greatly improved this write-up.

References

1. A. Beimel, M. Franklin. Reliable communication over partially authenticated networks. *Theoretical Computer Science*, 220:185–210, 1999.
2. A. Beimel, L. Malka. Efficient reliable communication over partially authenticated networks. In *the 22nd PODC*, pages 233–242, 2003.
3. M. Ben-Or, S. Goldwasser, A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *the 20th STOC*, pages 1–10, 1988.
4. M. Bläser, A. Jakoby, M. Liśkiewicz, B. Manthey. Private computation – k-connected vs. 1-connected networks. In *CRYPTO 2002*, vol. 2442 of *LNCS*, pages 194–209, 2002.
5. M. Bläser, A. Jakoby, M. Liśkiewicz, B. Manthey. Privacy in non-private environments. In *ASIACRYPT 2004*, vol. 3329 of *LNCS*, pages 137 – 151. 2004.
6. B. Bollobás. *Modern Graph Theory*. 1998.
7. D. Chaum, C. Crépeau, I. Damgård. Multiparty unconditionally secure protocols. In *the 20th STOC*, pages 11–19, 1988.
8. B. Chor, E. Kushilevitz. A zero-one law for Boolean privacy. *SIDMA*, 4(1):36–47, 1991.
9. Y. Desmedt, Y. Wang. Secure communication in multicast channels: The answer to Franklin and Wright's question. *J. of Cryptology*, 14(2):121–135, 2001.
10. Y. G. Desmedt, Y. Wang. Perfectly secure message transmission revisited. In *EUROCRYPT 2002*, vol. 2332 of *LNCS*, pages 502–517. 2002.
11. D. Dolev. The Byzantine generals strike again. *J. of Algorithms*, 3:14–30, 1982.
12. D. Dolev, C. Dwork, O. Waarts, M. Yung. Perfectly secure message transmission. *J. of the ACM*, 40(1):17–47, 1993.
13. C. Dwork, D. Peleg, N. Pippenger, E. Upfal. Fault tolerance in networks of bounded degree. *SIAM J. on Computing*, 17(5):975–988, 1988.
14. M. J. Fischer, N. A. Lynch, M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.
15. M. Franklin, R. N. Wright. Secure communication in minimal connectivity models. *J. of Cryptology*, 13(1):9–30, 2000.
16. M. Franklin, M. Yung. Secure hypergraphs: privacy from partial broadcast. In *the 25th STOC*, pages 36–44, 1993.

[3] This is a technical requirement as we can replace such non-curious neighbors by a new vertex holding the inputs of the two neighbor.

17. O. Goldreich, S. Goldwasser, N. Linial. Fault-tolerant computation in the full information model. In *the 32nd FOCS*, pages 447–457, 1991.
18. A. Jakoby, M. Liskiewicz, R. Reischuk. Private computations in networks: Topology versus randomness. In *the 20th STACS*, vol. 2607 of *LNCS*, pages 121–132. 2003.
19. M. V. N. A. Kumar, P. R. Goundan, K. Srinathan, C. Pandu Rangan. On perfectly secure communication over arbitrary networks. In *the 21st PODC*, pages 193–202, 2002.
20. E. Kushilevitz. Privacy and communication complexity. *SIDMA*, 5(2):273–284, 1992.
21. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, 1997.
22. K. Menger. Allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.
23. T. Rabin, M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *the 21st STOC*, pages 73–85, 1989.
24. H. M. Sayeed, H. Abu-Amara. Efficient perfectly secure message transmission in synchronous networks. *Information and Computation*, 126:53–61, 1996.
25. K. Srinathan, V. Vinod, C. Pandu Rangan. Efficient perfectly secure communication over synchronous networks. In *the 22nd PODC*, pages 252–252, 2003.
26. K. Srinathan, V. Vinod, C. Pandu Rangan. Optimal perfectly secure message transmission. In *CRYPTO 2004*, vol. 3152 of *LNCS*, pages 545 – 561, 2004.
27. E. Upfal. Tolerating a linear number of faults in networks of bounded degree. *Information and Computation*, 115(2):312–320, 1994.

Traffic Grooming on the Path*

Jean-Claude Bermond[1], Laurent Braud[2], and David Coudert[1,**]

[1] Mascotte Project, CNRS/I3S/INRIA – 2004 route des Lucioles – B.P. 93 – F-06902 Sophia-Antipolis Cedex – France
[2] ENS-Lyon – 46 allée d'Italie – F-69364 Lyon Cedex 07 – France

Abstract. In a WDM network, routing a request consists in assigning it a route in the physical network and a wavelength. If each request uses at most $1/C$ of the bandwidth of the wavelength, we will say that the grooming factor is C. That means that on a given edge of the network we can groom (group) at most C requests on the same wavelength. With this constraint the objective can be either to minimize the number of wavelengths (related to the transmission cost) or minimize the number of Add Drop Multiplexer (shortly ADM) used in the network (related to the cost of the nodes). Here we consider the case where the network is a path on N nodes, P_N. Thus the routing is unique. For a given grooming factor C minimizing the number of wavelengths is an easy problem, well known and related to the load problem. But minimizing the number of ADM's is NP-complete for a general set of requests and no results are known. Here we show how to model the problem as a graph partition problem and using tools of design theory we completely solve the case where $C = 2$ and where we have a static uniform all-to-all traffic (requests being all pairs of vertices).

1 Introduction

Traffic grooming is the generic term for packing low rate signals into higher speed streams (see the surveys [13, 22, 24]). By using traffic grooming, one can bypass the electronics in the nodes for which there is no traffic sourced or destinated to it. Typically, in a optical network using wavelength division multiplexing (WDM), instead of having one SONET Add Drop Multiplexer (shortly ADM) on every wavelength at every node, it may be possible to have ADMs only for the wavelength used at that node (the other wavelengths being optically routed without electronic switching). More precisely, in SONET networks, the bandwidth offered by a wavelength (typically 2.5 or 10 Gbits/sec.) is shared by several low speed streams. For instance, an OC-48 corresponds to a bandwidth of 2.5Gbits/sec is a container for 4 OC-12, each corresponding to a 655Mbits/sec

* This work has been partially funded by European projects FET CRESCCO and COST 293 GRAAL, and has been done in the CRC CORSO with France Telecom.
** Correspondig author: David.Coudert@sophia.inria.fr

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 34–48, 2005.

stream. In order to managed those bitstream, an ADM is to be placed each time a stream is added or dropped from a wavelength.

In the past many papers on WDM networks had for objective to minimize the transmission cost and in particular the number of wavelengths to be used [8, 1, 11], recent research has focused on reducing the total number of ADMs used in the network, trying to minimize it.

Here, we consider the particular case of paths (the routing is unique) with static uniform all-to-all traffic (requests being all pairs of vertices).

To each request $\{i, j\}$ routed on the path from i to j, we want to assign a wavelength in such a way that at most C requests use the same wavelength on a given edge of the path. Equivalently, each request uses $1/C$ of the bandwidth of the wavelength. C is called the *grooming ratio* (or *grooming factor*). For example, if the request from i to j is one OC-12 and a wavelength can carry an OC-48, the grooming factor is 4. Given the grooming ratio C and the length N of the path, the objective is to minimize the total number of (SONET) ADMs used, denoted $A(P_N, C)$, and so reducing the network cost by eliminating as many ADMs as possible from the "no grooming case".

Figure 1 shows how to groom requests for a grooming factor $C = 2$ and a path P_N with $N = 3, 7, 9$ vertices. For $N = 7$ we have 21 requests. So, a priori, if we give one wavelength to each request we need 42 ADMs. Using the same wavelength for disjoint requests (case $C = 1$) we will see after that 33 ADMs suffice. Indeed two requests may share an ADM if they have a common extremity. For $C = 2$ we will see that the construction given in Figure 1 is optimal and use 20 ADMs (note that 4 requests share the same ADM in vertex 3).

To the best of our knowledge, the problem for paths has only been studied in [10] where it has been proved NP-complete for a general set of requests and no other results are known. Other topologies have also been considered and in

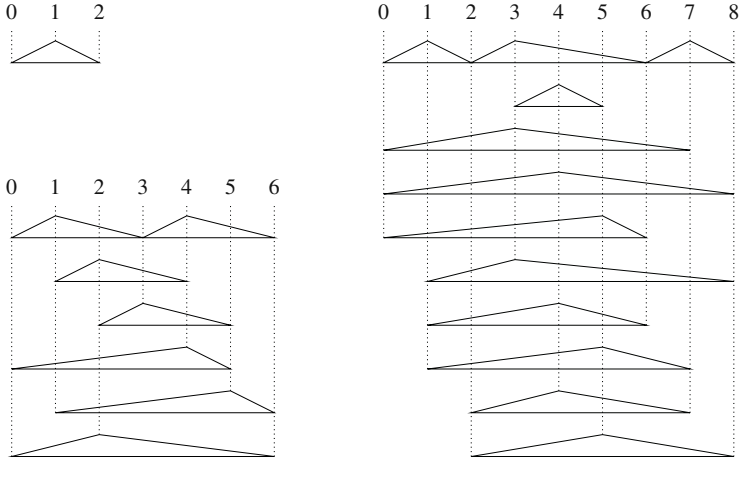


Fig. 1. Constructions for $N = 3, 7$ and 9

particular unidirectional rings primarily in the context of variable traffic requirements [6, 12, 17, 25, 27], but the case of fixed traffic requirements has served as an important special case [2, 3, 4, 5, 13, 15, 16, 19, 20, 22, 26, 28].

In this paper we model the grooming problem on the path as a graph partition problem. Then, we show how a greedy algorithm gives a solution for $C = 1$ and any set of requests. Thus, using tools of design theory, we determine exactly the number of ADMs in the case $C = 2$ for the all-to-all set of requests.

2 Modelization

Here we are given a physical graph and a set of requests. The physical graph will be the path P_N with vertex set $V = \{0, 1, 2, \ldots, N-1\}$ and where the edges are the pairs $\{i, i+1\}$, $0 \leq i \leq N-2$.

The set of requests I is a set of pairs $\{u, v\}$ that we model by a graph $G = (V, E)$ where each edge $e = \{u, v\}$ is associated to the request $\{u, v\}$. Each request is routed along the unique subpath from u to v and we associate to it a wavelength w.

For a subgraph B of requests of G, we define the load of an edge $e = \{i, i+1\}$ of P_N, $L(B, e)$, as the number of requests which are routed through e, that is the number of edges $\{u, v\}$ of B such that $u \leq i < v$.

Now let $B_w = (V_w, E_w)$ be the subgraph of G containing all requests carried by wavelength w. The fact that the grooming ratio is C can be expressed as $L(B_w, e) \leq C$ for each edge e of P_N. The number of ADMs used for wavelength w is nothing else than $|V_w|$.

So the problem corresponds to partition the edges of G (set of requests) into subgraphs B_w (set of requests with wavelength w) such that $L(B_w, e) \leq C$.

It is straightforward to see that minimizing the number W of wavelengths needed to route all requests is equivalent to minimize the number of subgraphs in the partition. Furthermore this is an easy problem since the load $L(G, e)$ is easy to compute. For example if G is the complete graph, $L(G, \{i, i+1\}) = (i+1)(N-i-1)$. If $L_{\max}(G)$ is the maximum load over all the edges, $L_{\max}(G) = \max_{e \in P_N} L(G, e)$, then we need at least $\frac{L_{\max}(G)}{C}$ wavelengths and we can assign them in a greedy way. For the complete graph, the number of wavelengths is therefore:

Proposition 1. *For the all-to-all set of requests on the path P_N and grooming ratio C, the minimum number of wavelength needed is $\left\lceil \frac{N^2-\epsilon}{4C} \right\rceil$, where $\epsilon = 1$ when N is odd and 0 otherwise.*

Proof. We have $L_{\max}(K_N) = \max_{e \in P_N} L(K_N, e) = \max_{\{i,i+1\}=e \in P_N} (i+1)(N-i-1) = \left\lceil \frac{N^2-\epsilon}{4} \right\rceil$, where $\epsilon = 1$ when N is odd and 0 otherwise.

Here our objective is to minimize the number of ADMs, that is the sum of the number of vertices in the B_w. Thus the problem can be formalized as follows:

Problem 1 (Grooming problem on the path).
Inputs : a path P_N, a grooming ratio C and a set of requests I modeled by the graph $G = (V, E)$
Output : a partition of the edges of G into subgraphs $B_w = (V_w, E_w)$, $w = 1, \ldots, W$, such that $\text{load}(B_w, e) \leq C$ for each edge e of P_N
Objective minimize $\sum_{1 \leq w \leq W} |V_w|$

Here we mainly consider $G = K_N$ and, following [4], we will denote $A(P_N, C)$ the optimal number of ADMs for a grooming ratio C and all-to-all set of requests on the path.

We have formalized the problem in its undirected version, but for paths it is the same for directed or symmetric directed versions. Indeed, if we consider a dipath $\overrightarrow{P_N}$ where the arcs are from i to $i+1$, and if the requests are the couples (u, v), with $u < v$, the problem is exactly the same. If we consider a symmetric dipath P_N^* with arcs $(i, i+1)$ and $(i+1, i)$ and the requests are the couples (u, v), we can split the problem into 2 disjoint subproblems, one with the dipath $\overrightarrow{P_N}$ oriented from 0 to $N-1$ with all requests (u, v) with $u < v$, and the second on the dipath $\overleftarrow{P_N}$ oriented from $N-1$ to 0 with requests (u, v) with $v < u$.

To the best of our knowledge, this problem has only been studied in [10] where it has been proved NP-complete, and no other results are known. However, the grooming problem for rings has been extensively studied. For example in [4] we have shown that the grooming problem on the unidirectional ring can be formalized as follows:

Problem 2 (Grooming problem on the cycle).
Inputs : a number of nodes N and a grooming ratio C
Output : a partition of the edges of K_N into subgraphs $B_w = (V_w, E_w)$, $w = 1, \ldots, W$, such that $|E_w| \leq C$
Objective minimize $\sum_{1 \leq w \leq W} |V_w|$

We denote $A(C_N, C)$ the optimal number of ADMs for a grooming ratio C and all-to-all set of requests on the unidirectional ring.

Note that in Problem 2, for the ring, it is supposed that the two requests (u, v) and (v, u) are assigned to the same wavelength (using thus $1/C$ of the capacity of the wavelength). Clearly, a bound on the number of ADMs for unidirectional ring gives a bound for our problem, but there might be very different (for example $A(C_3, 2) = 5$ but $A(P_3, 2) = 3$) due to capacity constraints.

In fact, the problem for unidirectional rings corresponds to the problem of path "*with erasure*" [10]. In this model a request (u, v) uses $1/C$ of the bandwidth on the whole path and not only on the subpath between u and v. The "load condition" becomes: there are at most C requests in any subgraph B_w which is exactly the constraint of Problem 2.

We will show in the next section that the grooming problem on the path for $C = 1$ and general instances can be solved polynomially, which is not the case on the ring (in the erasure model) [23, 25, 14].

3 Grooming Ratio $C = 1$

When the grooming ratio is equal to 1, the grooming problem on the path can be solved optimally for any set of requests in polynomial time. We prove this in Theorem 1 and give the exact number of ADMs in the all-to-all case in Corollary 1.

Theorem 1. $A(P_N, G, 1) = \sum_{i=0}^{N-1} \max\left\{d_G^-(i), d_G^+(i)\right\}$.

Proof. The lower bound is simple since in each node i of the path P_N we can not do better than sharing an ADM between a request ending in this node, that is a request $\{u, i\}$ with $u < i$, and a request starting from it, that is $\{i, v\}$ with $i < v$. Thus $A(P_N, G, 1) \geq \sum_{i=0}^{N-1} \max\left\{d_G^-(i), d_G^+(i)\right\}$.

Now, note that it is always possible to put a request ending in node i and a request starting from i in a same subgraph. Thus we can form the subgraphs using a greedy process: scan the nodes of the path from 0 to $N-2$ and add to each subgraph containing a request ending in i a requests starting from i (if any left), and then create a new subgraph for each remaining request that start from i (if any). So, in each node i, we will use $\max\left\{d_G^-(i), d_G^+(i)\right\}$ ADMs and so the lower bound is attained.

Finally, one may remark that this process will create more subgraphs than necessary, but we can merged two subgraphs if they contains disjoint requests. Doing so we will use the optimal number of subgraphs.

Corollary 1. $A(P_N, 1) = \frac{3N^2-2N-\epsilon}{4}$, *where* $\epsilon = 1$ *when* N *is odd and 0 otherwise .*

A simple construction is the following. First, one can easily check that $A(P_2, 1) = 2$ and $A(P_3, 1) = 5$. Then let the vertices of P_N be $0, 1, \ldots, N-1$, arrange them in this order, and suppose that $A(P_N, 1) = (3N^2 - 2N - \epsilon)/4$, where $\epsilon = 1$ when N is odd and 0 otherwise. Let now the vertices of P_{N+2} be $x, 0, 1, \ldots, N-1, y$ and arrange them in this order. The subgraphs of the partition of K_{N+2} will be: the N subgraphs B_j, $0 \leq j \leq N-1$, each of them containing the edges $\{x, j\}$ and $\{j, y\}$, and so $|V(B_j)| = 3$; the subgraph B_N which contains only the edge $\{x, y\}$, and so $|V(B_0)| = 2$; and the subgraphs of the partition of K_N. So altogether the partition of K_{N+2} contains $2 + 3N + (3N^2 - 2N - \epsilon)/4 = \left(3(N+2)^2 - 2(N+2) - \epsilon\right)/4$, where $\epsilon = 1$ when N is odd and 0 otherwise.

When the grooming ratio is $C \geq 2$, the problem is NP-complete and difficult to approximate for general instance. In particular, when the grooming ratio is equal to $C = 2$, this problem is similar to partition the edges of G into the maximum number of K_3 (see [9, 18]), although such partition only provides an upper bound of the total number of ADMs (two K_3 may share an ADM). However, for $G = K_N$ we will give in the next sections the exact number of ADMs for $C = 2$.

4 Lower Bounds

Consider a valid construction for Problem 1 and let a_p denote the number of subgraphs of the partition with exactly p nodes, A the number of ADMs, and W the number of subgraphs of the partition. We have the following equalities:

$$A = \sum_{p=2}^{N} p a_p \qquad (1) \qquad \sum_{p=2}^{N} a_p = W \qquad (2) \qquad \sum_{w=1}^{W} |E_w| = |E| \qquad (3)$$

In the particular case where $G = K_N$ we know by Proposition 1 that $W \geq \left\lceil \frac{N^2-\epsilon}{4C} \right\rceil$, where $\epsilon = 1$ when N is odd and 0 otherwise, and we have $E = \frac{N(N-1)}{2}$.

To obtain accurate lower bounds we need to bound the value of $|E_w|$ for a graph with $|V_w| = p$ vertices, satisfying the load constraint. Let $\gamma(C, p)$ be this maximum number of edges. Equations 2 and 3 becomes

$$\sum_{p=2}^{N} a_p \geq \left\lceil \frac{N^2-\epsilon}{4C} \right\rceil \qquad (4) \qquad \sum_{p=2}^{N} a_p \gamma(C,p) \geq \frac{N(N-1)}{2} \qquad (5)$$

In what follows we will restrict ourselves to the case $C = 2$, which is already non immediate and for which we have been able to obtain exact values. To obtain the right lower bounds when N is even, we need to determine $\gamma(2, p, 2h)$ which is the maximum number of edges of a graph B with p vertices with at least $2h$ vertices of odd degree and such that $L(B, e) \geq 2$ for each edge of P_N. Note that $\gamma(2, p) = \gamma(2, p, 0)$.

We will denote by $G + H$ the graph obtained by merging the right most node of G with the left most node of H.

Lemma 1. $\gamma(2, p, 2h) = \left\lfloor \frac{3p-3-h}{2} \right\rfloor$

Proof. We prove the lemma by induction. It is true for $p = 2$ as a graph with two vertices has at most one edge. In that case $h = 1$ and we have equality. For $p = 3$ the maximum number of edges is 3, obtained with a K_3, and there is equality for $h = 0$. With $h = 2$, the graph has at most 2 edges and the equality is attained with a P_3. Similarly for $p = 4$, the graph has at most 4 edges. Let the vertices be $\{a, b, c, d\}$ with $a < b < c < d$. For $h = 0$ the equality is obtained by the graph C_4 consisting of the 4 edges $\{a, b\}$, $\{b, c\}$, $\{c, d\}$ and $\{a, d\}$, and for $h = 1$ equality is attained by the graph consisting of an edge joined by a vertex to a K_3 more precisely the 4 edges $\{a, b\}$, $\{b, c\}$, $\{c, d\}$ and $\{b, d\}$.

Now consider a graph B with p vertices and $2h$ vertices of odd degree. Let $m(B)$ be the number of edges of B, and let u_0 be the first vertex (in the order of the path).

1. If u_0 has degree 1, $B - \{u_0\}$ has at least $2h - 2$ vertices of degree 1 and therefore $m(B) \leq \gamma(2, p-1, 2h-2) + 1 = \left\lfloor \frac{3p-3-h}{2} \right\rfloor$
2. If u_0 is of degree 2, let u_1 and u_2 be the 2 neighbors of u_0, with $u_0 < u_1 < u_2$. As $L(B, \{u_1 - 1, u_1\}) \leq 2$ there is no edge $\{u, u_1\}$ with $u <$

u_1, and as $L(B, \{u_1, u_1+1\}) \leq 2$ there is at most one edge $\{u_1, v\}$ with $v > u_1$.

(a) If there is no edge $\{u_1, v\}$, the graph obtained from B by deleting u_0 and u_1 has at least $2h-2$ vertices of odd degree and so $m(B) \leq \gamma(2, p-2, 2h-2) = \left\lfloor \frac{3p-4-h}{2} \right\rfloor$.

(b) If there is an edge $\{u_1, v_1\}$ 3 subcases can appear.
   i. either $v_1 = u_2$ and the graph obtained from B by deleting u_0 and u_1 (and therefore the K_3 $\{u_0, u_1, v_1\}$) has the same number of vertices of odd degree as B and so $m(B) \leq \gamma(2, p-2, 2h) = \left\lfloor \frac{3p-3-h}{2} \right\rfloor$.
   ii. or $v_1 < u_2$. Due to the load constraint there is no edge $\{u, v_1\}$ with $u < v_1$ and at most one edge $\{v_1, v\}$ with $v_1 < v$. The graph obtained from B by deleting u_0, u_1, v_1 has at least $2h-2$ vertices of odd degree and 3 or 4 edges less than B. So $m(B) \leq \gamma(2, p-3, 2h) = \left\lfloor \frac{3p-3-h}{2} \right\rfloor$.
   iii. or $v_1 > u_2$ we do the same reasoning by deleting from B the vertices u_0, u_1, u_2 and we obtain $m(B) = \left\lfloor \frac{3p-3-h}{2} \right\rfloor$.

So in all cases the bound is proved. Furthermore a careful analysis indicates when the bound is attained. An optimal $(p, 2h)$ can be obtained either by adding an edge joined to a vertex of even degree of a $(p-1, 2h-2)$ optimal graph (case 1); or by adding two edges $\{a, b\}$ and $\{a, c\}$ with $a < b < c$, c being a vertex of even degree of an optimal $(p-2, 2h-2)$ graph (case 2.a); or by adding a K_3 joined to a vertex of an optimal $(p-2, 2h)$ graph (case 2.b.i); or by adding a C_4 joined to a vertex of an optimal $(p-3, 2h)$ graph (careful analysis of case 2.b.iii).

In particular when p is odd and $h = 0$, the optimal graph is unique and consists of a sequence of K_3's sharing two by two a vertex $(K_3 + K_3 + \cdots + K_3)$.

For any h, equality is attained with the graph consisting of K_3s and h edges merged in the following way $e + K_3 + e + K_3 + \cdots + K_3 + e + K_3 + K_3 + \cdots + K_3$.

Theorem 2. *$A(P_N, 2) \geq \left\lceil \frac{11N^2-8N-3}{24} \right\rceil$ when N is odd, and when N is even $A(P_N, 2) \geq \left\lceil \frac{N(N-1)}{3} + \left\lceil \frac{N^2}{8} \right\rceil + \frac{N}{6} \right\rceil$.*

Proof. By Lemma 1 we know that $|E_w| \leq \gamma(2, p, 2h) = \frac{3p_w-3-h_w}{2}$ for a B_w with p_w vertices and $2h_w$ vertices with odd degree. So

$$\sum_{w=1}^{W} |E_w| \leq \sum_{p=2}^{N} \frac{3p-3}{2} a_p - \sum_{w=1}^{W} \frac{h_w}{2} \tag{6}$$

If N is odd, $\sum_{w=1}^{W} h_w$ can be equal to 0, but when N is even all vertices of K_N being of odd degree, $\sum_{w=1}^{W} 2h_w \geq N$. So Equations 1, 4 and 5 becomes

$$A = \sum_{p=2}^{N} p a_p \qquad (7) \qquad \sum_{p=2}^{N} a_p \geq \left\lceil \frac{N^2 - \epsilon}{8} \right\rceil \qquad (8)$$

$$\sum_{p=2}^{N} \frac{3p-3}{2} a_p - (1-\epsilon)\frac{N}{4} \geq \frac{N(N-1)}{2} \qquad (9)$$

Thus Equation 9 become

$$\sum_{p=2}^{N} 3p a_p \geq N(N-1) + 3\sum_{p=2}^{N} a_p + (1-\epsilon)\frac{N}{2} \qquad (10)$$

and so,
$$A(P_N, 2) \geq \frac{N(N-1)}{3} + \left\lceil \frac{N^2 - \epsilon}{8} \right\rceil + (1-\epsilon)\frac{N}{6} \qquad (11)$$

When N is odd, we have $\epsilon = 0$ and so $A(P_N, 2) \geq \frac{11N^2 - 8N - 3}{24}$, and when N is even, we have $\epsilon = 1$ and so $A(P_N, 2) \geq \left\lceil \frac{N(N-1)}{3} + \left\lceil \frac{N^2}{8} \right\rceil + \frac{N}{6} \right\rceil$

5 Constructions for $C = 2$

5.1 3-GDD

Let $v_1, v_2, \ldots, v_l$ be non negative integers; the *complete multipartite graph with group sizes* $v_1, v_2, \ldots, v_l$ is defined to be the graph with vertex set $V_1 \cup V_2 \cup \cdots \cup V_l$ where $|V_i| = v_i$, and two vertices $u \in V_i$ and $v \in V_j$ are adjacent if $i \neq j$. Using terminology of Design Theory, the graph of type $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_l^{\alpha_l}$ will be the complete multipartite graph with α_i groups of size p_i. The existence of a partition of this multipartite graph into K_k is equivalent to the existence of a *k-GDD (Group Divisible Design) of type* $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_l^{\alpha_l}$.

Here we are interested in the existence of 3-GDD's, that is partitions into K_3's.

Theorem 3 (Existence of a 3-GDD (see [7])). *There exists a 3-GDD of type* $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_l^{\alpha_l}$ *if and only if (i) each node of the complete multipartite graph has even degree, and (ii) the number of edges is a multiple of 3.*

It follows that when $N \equiv 1$ or $3 \pmod 6$, K_N can always be partitioned into K_3. Various constructions are explained in [21]. One can found in [7] a collection of multipartite graphs for which there exists a 3-GDD.

5.2 Constructions for Small Values of N

We have reported in the following table the number of ADMs and the number of subgraphs of optimal constructions for some small cases. The most important constructions are given in Section A.

N	2	3	4	5	6	7	8	9	10	11	12	13	16	17	20
$A(P_N, 2)$	2	5	7	10	16	20	28	34	45	52	64	73	115	127	180
Nb subgraphs	1	1	2	3	6	6	8	10	13	15	18	20	32	36	50

5.3 Constructions for Odd Values

In this section we will show that the lower bound is attained for odd values and we will prove it by induction. Note that to have equality, an optimal solution has to contains the minimum number of subgraphs, that is $\left\lceil \frac{N^2-1}{8} \right\rceil$. If $N \equiv 1$ or $3 \pmod 6$, any subgraph of the decomposition with p nodes has exactly $\frac{3p-3}{2}$ edges, which implies p odd and no vertices of odd degree. So the subgraphs of the decomposition are of the form $K_3 + K_3 + \cdots + K_3$. If $N \equiv 5 \pmod 6$, an optimal decomposition consists of K_3's and one C_4, some of them being merged together.

Theorem 4 (1.26 page 190 of [7]). *Let u and v be positive integer with $v \leq u$. Then a 3-GDD of type $u^1v^1 1^u$ exists if and only if $(u, v) \equiv (1, 1)$, $(3, 1)$, $(3, 3)$, $(3, 5)$, $(5, 1)$ $(mod\ (6, 6))$.*

Corollary 2. *Given u and v satisfying the condition of Theorem 4 and an optimal construction for both u and v, we can build an optimal construction for $N = 2u + v$.*

Proof. Let the nodes of K_N be numbered from left to right $0, 1, \ldots, u-1, u, \ldots, u+v-1, \ldots, 2u+v-1 = N$ and let $A = \{0, 1, \ldots, u-1\}$, $B = \{u, u+1, \ldots, u+v-1\}$ and $C = \{u+v, u+v+1, \ldots, 2u+v-1\}$.

The 3-GDD of type $u^1v^1 1^u$ has $\frac{3u^2-u+4uv}{6}$ K_3, and we say that the K_3s are of type ABC or ACC or CCC depending of their number of nodes in A, B and C. There are uv K_3 of type ABC, $\frac{u(u-v)}{2}$ K_3 of type ACC and $\frac{u(v-1)}{6}$ K_3 of type CCC.

Note that as expected the number of subgraphs in the partition is $\frac{u^2-1}{8} + \frac{3u^2-u+4uv}{6} - \frac{u(v-1)}{6} = \frac{(2u+v)^2-1}{8}$.

Each node of A is the left most node of $v + \frac{u-v}{2} = \frac{u+v}{2}$ K_3 of type ABC or ACC. Since each node of A is the right most node of at most $\frac{u-1}{2}$ subgraphs of the partition of K_u, we can merged each subgraph with one K_3 and so we save $\frac{u^2-1}{8}$ ADMs.

Each node of C is the right most node of v K_3 of type ABC. It is also involved in $u - v$ K_3 of type ACC and in $\frac{u-1-(u-v)}{2} = \frac{v-1}{2}$ K_3 of type CCC. Thus we can merged each K_3 of type CCC with a K_3 of type ABC and so we save $\frac{u(v-1)}{6}$ more ADMs.

Note that since each node of B is the middle node of a K_3 of type $\{a, b, c\}$, we can not merge the subgraphs of the partition of K_v.

Finally, the construction use $\frac{3u^2-u+4uv}{2} + A(P_u, 2) - \frac{u^2-1}{8} - \frac{u(v-1)}{6} + A(P_v, 2) = \frac{3u^2-u+4uv}{2} + \frac{11u^2-8u-3}{24} - \frac{u^2-1}{8} - \frac{u(v-1)}{6} + \frac{11v^2-8v-3}{24} = \frac{11(2u+v)^2-8(2u+v)-3}{24}$, which is the lower bound.

Theorem 5. *When N is odd, $A(P_N, 2) = \left\lceil \frac{11N^2-8N-3}{24} \right\rceil$. Furthermore, the construction contains $\frac{N^2-1}{8}$ subgraphs.*

Proof. For $N = 3, 5, 7, 13, 17$ we give direct constructions in Lemmas 3, 5, 6, 9 and 11. For other values we will use Corollary 2 using induction on u.

- When $N = 12t + 1$, $t \geq 2$, let $u = 6t - 3$ and $v = 7$. Since $(6t - 3, 7) \equiv (3, 1) \pmod{6, 6}$, we can used Corollary 2.
- When $N = 12t + 3$, $t \geq 0$, we can use Corollary 2 with $u = 6t + 1$ and $v = 1$
- When $N = 12t + 5$, $t \geq 3$, we can use Corollary 2 with $u = 6t - 3$ and $v = 11$, and for $N = 29$ we can used Corollary 2 with $u = 11$ and $v = 7$
- When $N = 12t + 7$, $t \geq 0$, we can used Corollary 2 with $u = 6t + 3$ and $v = 1$
- When $N = 12t + 9$, $t \geq 0$, we can used Corollary 2 with $u = 6t + 3$ and $v = 3$.
- When $N = 12t + 11$, $t \geq 1$, we can used Corollary 2 with $u = 6t + 3$ and $v = 5$. Finally, we can also use Corollary 2 for $N = 11$ with $u = 5$ and $v = 1$

5.4 Construction for Even Values

In view of the lower bound, an optimal partition will have exactly $\left\lceil \frac{N^2}{8} \right\rceil$ subgraphs and each vertex will appear with odd degree and otherwise the value $\frac{3p-3}{2}$ is attained. So we will have mainly K_3's, plus $\frac{N}{2}$ graphs $K_3 + e$ (except for some congruence classes where one edge is isolated) some of these K_3's or $K_3 + e$ being merged together.

Lemma 2. *There exists a 3-GDD of type* $(2u)^1(2v)^1 2^u$ *when* $u \geq v \geq 1$ *and* $u(v-1) \equiv 0 \pmod 3$.

Proof. To prove that, one has to check that all nodes have even degree (which is true) and that the total number of edges is a multiple of 3.

Since we have $4u^2 + 4uv + 4uv + 4\frac{u(u-1)}{2} = 6u^2 + 6uv + 2u(v-1)$ edges it remains to check that $u(v-1) \equiv 0 \pmod 3$.

Theorem 6. *When N is even,* $A(P_N, 2) = \left\lceil \frac{N(N-1)}{3} + \left\lceil \frac{N^2}{8} \right\rceil + \frac{N}{6} \right\rceil = \frac{11N^2-4N}{24} + \epsilon_N$, *where* $\epsilon_N = \frac{1}{2}$ *when* $N \equiv 2$ *or* $6 \pmod{12}$, $\epsilon_N = \frac{1}{3}$ *when* $N \equiv 4 \pmod{12}$, $\epsilon_N = \frac{5}{6}$ *when* $N \equiv 10 \pmod{12}$, *and 0 when* $N \equiv 0$ *or* $8 \pmod 6$. *Furthermore, the construction contains* $\left\lceil \frac{N^2}{8} \right\rceil$ *subgraphs.*

Proof. First of all, we know from Lemmas 3, 4, 7, 8, 10 and 12 that the theorem is true for $N = 2, 4, 8, 12, 16, 20$.

Now suppose that the result is true for $2u$ and $2v$, that is for $w = u$ or v,

$$A(P_{2w}, 2) = \left\lceil \frac{2w(2w-1)}{3} + \left\lceil \frac{4w^2}{8} \right\rceil + \frac{2w}{6} \right\rceil = \frac{44w^2 - 4w}{24} + \epsilon_w \tag{12}$$

where $\epsilon_w = \frac{1}{2}$ when $2w \equiv 2$ or $6 \pmod{12}$, $\epsilon_w = \frac{1}{3}$ when $2w \equiv 4 \pmod{12}$, $\epsilon_w = \frac{5}{6}$ when $2w \equiv 10 \pmod{12}$, and 0 otherwise. Furthermore, the construction use $\left\lceil \frac{4w^2}{8} \right\rceil$ subgraphs.

Let now $N = 4u + 2v$, where u and v are such that there exists a 3-GDD of type $(2u)^1(2v)^1 2^u$. Let also the nodes be $A, B, C_1, C_2, \ldots, C_u$ with $|A| = 2u$, $|B| = 2v$ and $|C_i| = 2$, $1 \leq i \leq u$, and let $C = \cup_{i=1}^{u} C_i$.

To simplify the notation, we say that an edge is of type CC if it has one node in C_i and another in C_j with $i \neq j$.

The 3-GDD of type $(2u)^1(2v)^1 2^u$ has $\frac{6u^2-2u+8uv}{3}$ K_3: $4uv$ of type ABC, $\frac{2u(2u-2v)}{2} = 2u(u-v)$ of type ACC and $\frac{2u(v-1)}{3}$ of type CCC.

We observe that each node of C is the right most node of $2v$ K_3 of type ABC and is involved in $2u - 2v$ K_3 of type ACC and $\frac{2u-2-(2u-2v)}{2} = v - 1$ K_3 of type CCC. Thus, we can merge each K_3 of type CCC with a K_3 of type ABC and so save $\frac{2u(v-1)}{3}$ ADMs. Furthermore, we can merged each edge $\left\{c_i^1, c_i^2\right\}$ such that $c_i^1, c_i^2 \in C_i$, $1 \leq i \leq u$, with a K_3 of type ABC or ACC and so save u more ADMs.

Each node of A is the left most node of $2v + \frac{2u-2v}{2} = u + v$ K_3 of type ABC or ACC and is the right most node of at most $\frac{2u-2}{2} + 1 = u$ subgraphs of the optimal construction for $2u$. Thus we can merged each subgraph and save $\left\lceil \frac{4u^2}{8} \right\rceil$ more ADMs.

By hypothesis we have

$$A(P_{2u}, 2) - \left\lceil \frac{4u^2}{8} \right\rceil = \left\lceil \frac{2u(2u-1)}{3} + \frac{2u}{6} \right\rceil = \left\lceil \frac{u(4u-1)}{3} \right\rceil = \frac{u(4u-1)}{3} + \alpha_u \quad (13)$$

where $\alpha_u = \frac{1}{3}$ when $u \equiv 2 \pmod 3$ and 0 otherwise.

Altogether the construction uses the following number of ADMs.

$$\begin{aligned} A(P_N, 2) \leq\ & A(P_{2u}, 2) - \left\lceil \tfrac{4u^2}{8} \right\rceil + A(P_{2v}, 2) + (6u^2 - 2u + 8uv) \\ & - \tfrac{2u(v-1)}{3} + 2u - u \qquad (14) \\ \leq\ & \frac{u(4u-1)}{3} + \alpha_u + \frac{44v^2 - 8v}{24} + \epsilon_v + \frac{18u^2 - u + 22uv}{3} \qquad (15) \\ \leq\ & \frac{11(4u+2v)^2 - 4(4u+2v)}{24} + \alpha_u + \epsilon_v \qquad (16) \end{aligned}$$

Now we have to check that $\alpha_u + \epsilon_v = \epsilon_N$ in all cases. For that, observe that the conditions of Lemma 2 are satisfied when $v = 1$ and when $v = 4$, assuming that $u \geq v \geq 1$. So we have reported in the following table all cases that satisfies the above construction.

N	condition	u	v	α_u	ϵ_v	$\alpha_u + \epsilon_v$	ϵ_N
$12t+2$	$t \geq 1$	$3t$	1	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$12t+4$	$t \geq 2$	$3t-1$	4	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$
$12t+6$	$t \geq 0$	$3t+1$	1	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$12t+8$	$t \geq 2$	$3t$	4	0	0	0	0
$12t+10$	$t \geq 0$	$3t+2$	1	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{5}{6}$	$\frac{5}{6}$
$12t+12$	$t \geq 1$	$3t+1$	4	0	0	0	0

Furthermore, the number of subgraphs in our construction for $N = 4u + 2v$ is equal to the number of K_3 of type ABC plus the number of K_3 of type ACC and plus the number of subgraphs in the construction for $2v$, that is $4uv + 2u(u-v) + \left\lceil \frac{4v^2}{8} \right\rceil = \left\lceil \frac{(4u+2v)^2}{8} \right\rceil$.

In conclusion, Theorem 6 is true for all even N.

Acknowledgment

Many thanks to C.J. Colbourn for his help in solving the case $N = 17$.

References

1. B. Beauquier, J-C. Bermond, L. Gargano, P. Hell, S. Pérennes, and U. Vaccaro. Graph problems arising from wavelength-routing in all-optical networks. In *IEEE Workshop on Optics and Computer Science*, Geneva, Switzerland, April 1997.
2. J-C. Bermond and S. Ceroi. Minimizing SONET ADMs in unidirectional WDM ring with grooming ratio 3. *Networks*, 41(2), February 2003.
3. J.-C. Bermond, C.J. Colbourn, A. Ling, and M.-L. Yu. Grooming in unidirectional rings : $K_4 - e$ designs. *Discrete Mathematics, Lindner's Volume*, 284(1-3):57–62, 2004.
4. J-C. Bermond and D. Coudert. Traffic grooming in unidirectional WDM ring networks using design theory. In *IEEE ICC*, Anchorage, Alaska, May 2003.
5. J-C. Bermond, D. Coudert, and X. Muñoz. Traffic grooming in unidirectional WDM ring networks: the all-to-all unitary case. In *IFIP ONDM*, pages 1135–1153, February 2003.
6. A. L. Chiu and E. H. Modiano. Traffic grooming algorithms for reducing electronic multiplexing costs in WDM ring networks. *IEEE/OSA Journal of Lightwave Technology*, 18(1):2–12, January 2000.
7. C.J. Colbourn and J.H. Dinitz, editors. *The CRC handbook of Combinatorial designs*. CRC Press, 1996.
8. D. Coudert and H. Rivano. Lightpath assignment for multifibers WDM optical networks with wavelength translators. In *IEEE Globecom*, Taipei, Taiwan, November 2002.
9. D Dor and M. Tarse. Graph decomposition is NP-complete: a complete proof of Holyer's conjecture. *SIAM Journal on Computing*, 26(4):1166–1187, 1997.
10. R. Dutta, S. Huang, and N. Rouskas. On optimal traffic grooming in elemental network topologies. In *Opticomm*, pages 13–24, Dallas, October 2003.
11. R. Dutta and N. Rouskas. A survey of virtual topology design algorithms for wavelength routed optical networks. *Optical Networks*, 1(1):73–89, January 2000.
12. R. Dutta and N. Rouskas. On optimal traffic grooming in WDM rings. *IEEE Journal of Selected Areas in Communications*, 20(1):1–12, January 2002.
13. R. Dutta and N. Rouskas. Traffic grooming in WDM networks: Past and future. *IEEE Network*, 16(6):46–56, November/December 2002.
14. L. Epstein and A. Levin. Better bounds for minimizing sonet adms. In *WAOA*, September 2004.
15. O. Gerstel, P. Lin, and G. Sasaki. Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings. In *IEEE Infocom*, pages 94–101, San Francisco, California, 1998.

16. O. Gerstel, R. Ramaswani, and G. Sasaki. Cost-effective traffic grooming in WDM rings. *IEEE/ACM Transactions on Networking*, 8(5):618–630, 2000.
17. O. Goldschmidt, D. Hochbaum, A. Levin, and E. Olinick. The SONET edge-partition problem. *Networks*, 41(1):13–23, 2003.
18. I. Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.
19. J.Q. Hu. Optimal traffic grooming for wavelength-division-multiplexing rings with all-to-all uniform traffic. *OSA Journal of Optical Networks*, 1(1):32–42, 2002.
20. J.Q. Hu. Traffic grooming in wdm ring networks: A linear programming solution. *OSA Journal of Optical Networks*, 1(11):397–408, 2002.
21. C.C. Lindner and C.A. Rodger. *Design Theory*. CRC Press, 1997. ISBN 0849339863.
22. E. Modiano and P. Lin. Traffic grooming in WDM networks. *IEEE Communications Magazine*, 39(7):124–129, July 2001.
23. M. Shalom and S. Zaks. A $10/7 + \epsilon$ approximation for minimizing the number of adms in sonet rings. In *IEEE BroadNets*, pages 254–262, October 2004.
24. A. Somani. Survivable traffic grooming in WDM networks. In D.K. Gautam, editor, *Broad band optical fiber communications technology – BBOFCT*, pages 17–45, Jalgaon, India, December 2001. Nirtali Prakashan. Invited paper.
25. P-J. Wan, G. Calinescu, L. Liu, and O. Frieder. Grooming of arbitrary traffic in SONET/WDM BLSRs. *IEEE Journal of Selected Areas in Communications*, 18(10):1995–2003, October 2000.
26. J. Wang, W. Cho, V. Vemuri, and B. Mukherjee. Improved approches for cost-effective traffic grooming in WDM ring networks: Ilp formulations and single-hop and multihop connections. *IEEE/OSA Journal of Lightwave Technology*, 19(11):1645–1653, November 2001.
27. X. Yuan and A. Fulay. Wavelength assignment to minimize the number of SONET ADMs in WDM rings. In *IEEE ICC*, New York, April 2002.
28. X. Zhang and C. Qiao. An effective and comprehensive approach for traffic grooming and wavelength assignment in SONET/WDM rings. *IEEE/ACM Transactions on Networking*, 8(5):608–617, 2000.

A Small Cases

Remark that all the subgraphs that we consider in the constructions satisfies $L(B_w, e) \leq 2$. It is clear for a K_3 $\{u, v, w\}$ where we suppose $u < v < w$. For an edge $\{t, u\}$ glued with the K_3 $\{u, v, w\}$, we suppose that $t < u < v < w$.

Lemma 3. *$A(P_2, 2) = 2$ and $A(P_3, 2) = 3$.*

Lemma 4. *$A(P_4, 2) = 7$.*

Proof. Let the vertices of P_4 be $\mathbb{Z}_4$. The first subgraph contains the K_3 $\{1, 2, 3\}$ plus the edge $\{0, 1\}$, and the second subgraph contains the two edges $\{0, 2\}$ and $\{0, 3\}$.

Lemma 5. $A(P_5, 2) = 10$.

Proof. Let the vertices of P_5 be $\mathbb{Z}_5$. The graphs of the decomposition are the 2 K_3 $\{0, 2, 4\}$ and $\{0, 1, 3\}$ plus the subgraph B_3 containing the 4 edges $\{1, 2\}$, $\{2, 3\}$, $\{3, 4\}$ and $\{1, 4\}$. This construction fit the lower bound.

Lemma 6. $A(P_7, 2) = 20$

Proof. Let the vertices of P_7 be $\mathbb{Z}_7$, that is $0, 1, 2, 3, 4, 5, 6$. The construction is obtained using the partition of K_7 into the 7 K_3 $\{i, i + 1, i + 3\}$, indices being taken modulo 7, and the remark that the 2 K_3 $\{0, 1, 3\}$ and $\{3, 4, 6\}$ fit in a same subgraph. This construction use 20 ADMs and according to Theorem 2 we have $A(P_7, 2) \geq 20$.

Lemma 7. $A(P_8, 2) = 28$

Proof. Let the nodes be $a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2$. We have 4 groups of 2 consecutive nodes and we use a 3-GDD of type 2^4. Our construction consist on the 4 K_3 $\{a_2, b_2, c_2\}$, $\{b_1, c_2, d_1\}$, $\{a_1, c_2, d_2\}$ and $\{a_1, b_2, d_1\}$ plus the 4 $K_3 + e$ $\{a_1, a_2\} + \{a_2, b_1, d_2\}$, $\{b_1, b_2\} + \{b_2, c_1, d_2\}$, $\{a_1, b_1, c_1\} + \{c_1, c_2\}$ and $\{a_2, c_1, d_1\} + \{d_1, d_2\}$. This construction use 28 ADMs.

Lemma 8. $A(P_{12}, 2) = 64$

Proof. Let the nodes of P_{12} be $a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2, e_1, e_2, f_1, f_2$ and arrange them in this order.

Our construction consist on the 2 subgraphs (union of K_3) $\{a_1, b_1, c_1\} + \{c_1, e_2, f_1\}$ and $\{a_2, c_1, d_2\} + \{d_2, e_1, f_2\}$, plus the 6 $K_3 + e$ $\{a_1, a_2\} + \{a_2, b_2, f_1\}$, $\{b_1, b_2\} + \{b_2, c_2, d_2\}$, $\{c_1, c_2\} + \{c_1, d_1, e_1\}$, $\{a_2, c_2, d_1\} + \{d_1, d_2\}$, $\{a_2, b_1, e_1\} + \{e_1, e_2\}$ and $\{a_1, d_2, f_1\} + \{f_1, f_2\}$, and plus the 10 K_3 $\{b_1, d_1, f_1\}$, $\{b_2, d_1, e_2\}$, $\{a_1, c_2, e_2\}$,$\{b_1, c_2, f_2\}$,$\{a_1, d_1, f_2\}$,$\{b_2, c_1, f_2\}$,$\{a_1, b_2, e_1\}$,$\{b_1, d_2, e_2\}$,$\{c_2, e_1, f_1\}$ and $\{a_2, e_2, f_2\}$. Altogether, we use $2 \times 5 + 6 \times 4 + 10 \times 3 = 64$ ADMs.

Lemma 9. $A(P_{13}, 2) = 73$

Proof. Let the vertices of P_{13} be $\mathbb{Z}_{13}$ and remark that K_{13} can be partitioned into the 26 K_3 $\{i, i + 1, i + 4\}$ and $\{i, i + 5, i + 7\}$, $i \in \mathbb{Z}_{13}$. Our construction consist on the subgraph $\{0, 1, 4\} + \{4, 5, 8\} + \{8, 9, 12\}$, plus the 3 subgraphs $\{i, i + 1, i + 4\} + \{i + 4, i + 5, i + 8\}$, $i = 1, 2, 3$, plus the 4 K_3 $\{j, j + 1, j + 4\}$, $j = 9, 10, 11, 12$, and plus the 13 K_3 $\{k, k + 5, k + 7\}$, $k \in \mathbb{Z}_{13}$. Altogether this construction use $7 + 3 \times 5 + 17 \times 3 = 73$ ADMs and according to Theorem 2 we have $A(P_{13}, 2) \geq 73$.

Lemma 10. $A(P_{16}, 2) = 115$

Proof. Let the vertices of P_{16} be $A \cup B \cup C$, where $A = \{a_0, a_1, a_2, a_3, a_4, a_5\}$, $B = \{b_0, b_1, b_2, b_3\}$ and $C = \{c_0, c_1, c_2, c_3, c_4, c_5\}$. Our construction is based on the existence of a 3-GDD of type $6^1 4^1 2^3$, which consist on 24 K_3 of type ABC, 6 K_3 of type ACC and 2 K_3 of type CCC, and by merging the 5 subgraphs of the decomposition of K_6 with K_3s of type ABC, the 2 K_3 of type CCC and the 3 edges $\{c_i, c_{i+1}\}$, $i = 0, 1, 2$, with K_3s of type ABC. Altogether this construction use 115 ADMs and the subgraphs of the decomposition are:

- The 4 graphs on 5 vertices $\{a_0, b_0, c_1\} + \{c_1, c_2, c_4\}$, $\{a_2, b_1, c_0\} + \{c_0, c_3, c_4\}$, $\{a_0, a_2, a_5\} + \{a_5, b_1, c_1\}$ and $\{a_1, a_3, a_5\} + \{a_5, b_2, c_2\}$, so 20 ADMs.
- The 4 $K_3 + e$ $\{a_2, b_3, c_0\} + \{c_0, c_1\}$, $\{a_1, b_0, c_2\} + \{c_2, c_3\}$, $\{a_0, b_2, c_4\} + \{c_4, c_5\}$ and $\{a_2, a_3\} + \{a_3, b_0, c_5\}$, so 16 ADMs
- The 2 graphs on 6 vertices $(2K_3 + e)$ $\{a_0, a_3, a_4\} + \{a_4, a_5\} + \{a_5, b_0, c_4\}$ and $\{a_0, a_1\} + \{a_1, a_2, a_4\} + \{a_4, b_0, c_0\}$, so 12 ADMs,
- The 21 K_3 $\{a_0, b_1, c_3\}$, $\{a_0, b_3, c_5\}$, $\{a_1, b_1, c_4\}$, $\{a_1, b_2, c_1\}$, $\{a_1, b_3, c_3\}$, $\{a_2, b_0, c_3\}$, $\{a_2, b_2, c_3\}$, $\{a_3, b_1, c_5\}$, $\{a_3, b_2, c_5\}$, $\{a_3, b_3, c_4\}$, $\{a_4, b_1, c_2\}$, $\{a_4, b_2, c_0\}$, $\{a_4, b_3, c_2\}$, $\{a_5, b_3, c_1\}$, $\{a_0, c_0, c_2\}$, $\{a_1, c_0, c_5\}$, $\{a_2, c_2, c_5\}$, $\{a_3, c_1, c_3\}$, $\{a_4, c_1, c_5\}$, $\{a_5, c_3, c_5\}$ and $\{b_0, b_2, b_3\}$, so 63 ADMs
- The star $\{b_0, b_1\} + \{b_1, b_2\} + \{b_1, b_3\}$, so 4 ADMs.

Lemma 11. $A(P_{17}, 2) = 127$

Proof. Let the vertices of P_{17} be $\mathbb{Z}_{17}$. The decomposition is based on the existence of a 3-GDD of type $3^2 5^1 3^2$ (which was kindly given to us by C.J. Colbourn) and the subgraphs are:

- The 9 graphs on 5 vertices (consisting of two K_3s with a common vertex, the one in the middle) $\{0, 1, 2\} + \{2, 5, 11\}$, $\{3, 4, 5\} + \{5, 13, 15\}$, $\{1, 4, 11\} + \{11, 12, 13\}$, $\{2, 4, 14\} + \{14, 15, 16\}$, $\{0, 5, 6\} + \{6, 11, 14\}$, $\{2, 3, 7\} + \{7, 11, 16\}$, $\{0, 4, 8\} + \{8, 11, 15\}$, $\{1, 5, 9\} + \{9, 13, 14\}$ and $\{0, 3, 10\} + \{10, 12, 14\}$, so altogether 45 ADMs.
- The 24 K_3s $\{4, 6, 12\}$, $\{1, 6, 13\}$, $\{2, 6, 15\}$, $\{3, 6, 16\}$ $\{1, 7, 12\}$, $\{4, 7, 13\}$, $\{3, 7, 15\}$, $\{0, 7, 14\}$ $\{2, 8, 12\}$, $\{3, 8, 13\}$, $\{1, 8, 16\}$, $\{5, 8, 14\}$ $\{3, 9, 12\}$, $\{4, 9, 15\}$, $\{2, 9, 16\}$, $\{0, 9, 11\}$ $\{2, 10, 13\}$, $\{1, 10, 15\}$, $\{4, 10, 16\}$, $\{5, 10, 11\}$ $\{1, 3, 14\}$, $\{0, 12, 15\}$, $\{0, 13, 16\}$ and $\{5, 12, 16\}$, so 72 ADMs.
- The 3 graphs decomposing the K_5 on $6, 7, 8, 9, 10$, the 3 K_3 $\{6, 8, 10\}$ and $\{6, 7, 9\}$ and the C_4 $\{7, 8, 9, 10\}$, so 10 more ADMs.

In summary our construction use 127 ADMs, the lower bound.

Lemma 12. $A(P_{20}, 2) = 180$

Proof. The construction is similar to the construction of Lemma 10 and use a 3-GDD of type $2^3 8^1 2^3$.

Range Augmentation Problems in Static Ad-Hoc Wireless Networks

Davide Bilò[1] and Guido Proietti[1,2]

[1] Dipartimento di Informatica, Università di L'Aquila,
Via Vetoio, 67010 L'Aquila, Italy
{davide.bilo, proietti}@di.univaq.it

[2] Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti",
IASI-CNR, Viale Manzoni 30, 00185 Roma, Italy

Abstract. Given a set V of n stations, a transmission power cost function $c : V \times V \mapsto \mathbb{R}^+ \cup \{+\infty\}$, an initial power assignment $p_0 : V \mapsto \mathbb{R}^+$, and a connectivity property π, a *range augmentation problem* consists of finding a minimum power augmentation assignment p such that $p_f(\cdot) = p_0(\cdot) + p(\cdot)$ satisfies property π. In this paper, we focus on the problem of biconnecting an already existing connected network, to make it resilient to the failure of either a wireless link or a station. For these problems we give a $\mathcal{H}_n^2$-approximation greedy algorithm (where $\mathcal{H}_n$ is the n-th harmonic number) after proving that they are both not approximable within $(1 - o(1)) \ln n$, unless $\mathsf{NP} \subset \mathsf{DTIME}(n^{\mathcal{O}(\log \log n)})$, even when c is a distance cost function restricted to three power levels, or it is a distance cost function and p_0 induces a tree network. Moreover, we prove that both problems remain APX-hard even if the initial power assignment is uniform. In this latter scenario, we finally show that any λ-approximation algorithm for the corresponding problem in wired networks, is a 2λ-approximation algorithm for the wireless case.

Keywords: Radio Networks, Connectivity Augmentation, Network Survivability, Approximation Algorithms.

1 Introduction

In the last decades, human interest towards wireless telecommunication networks has considerably grown-up. This is essentially due to the fact that when the communication goes partially or totally via ether, then most of the rigidity of the classic wired communication model can be overcome. In particular, pure wireless networking requires no fixed backbone communication infrastructure, and thus hosts can exchange messages only through radio signals. This can be done either directly (*single-hop* model), or, more realistically, by traversing intermediate hosts (*multi-hop* model [19]). In this latter scenario, *static ad-hoc wireless networks* (or simply *radio networks*) are indisputably emerging as one of the most popular models [12, 22], thanks to their extreme versatility.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 49–64, 2005.

A radio network can be viewed as a collection of homogenous battery-operated *stations* (or *nodes*), i.e., stationary radio transmitters/receivers which can communicate with each other by sending/receiving radio signals. In practice, stations are usually equipped with omnidirectional antennas, and this is the model we assume for this paper. Each node transmits messages at a certain power level, but interferences from other transmissions and background noise may attenuate the signal. So the transmission power level of a station s affects its *transmission range*, i.e., the set of nodes s can directly send messages to. These two quantities are strictly related: the higher is the transmission power of a node, the larger will be its transmission range. For the purpose of avoiding interference problems and saving energy consumption, each node may vary its transmission power.

In a realistic setting, the power of a signal falls as $1/d^\alpha$, where d is the distance from the transmitter s and $\alpha \geq 1$ represents the *distance-power gradient* [21]. In particular, if the receiver t is at a distance $d(s,t)$ from s, and s transmits with power $p(s)$, then the power detected by t is $p(s)/d(s,t)^\alpha$. As one can assume that the signal sent from s can be decoded by the receiver t if the power detected by t is at least 1, then it is clear that $p(s)$ must be no less than $d(s,t)^\alpha$. Moreover, in any real setting $2 \leq \alpha \leq 4$ (usually $\alpha = 2$).

Survivable network design problem. Given an embedding network, a *network design problem* consists of finding an optimal subnetwork satisfying some connectivity property π, and that minimizes/maximizes some objective function. For instance, given an edge-weighted undirected graph, the cheapest connected network with respect to the sum of its edges is clearly a *minimum spanning tree*. Given current attention to the network reliability, one usually desires to build networks which are resilient to component crashes. A network with edge-connectivity k allows communication between each pair of nodes even if at most $k-1$ links fails. This definition straightforwardly extends to the vertex connectivity case.

Besides the problem of building-up a fault-tolerant network from scratch, sometimes one may be interested in strengthening the reliability of an already existing network. If this problem is already relevant for wired networks, it becomes more important for wireless networks. Indeed, interferences from other signals and environmental conditions (e.g., storms, rain, fog, etc.) make link crashes more frequent in wireless networks than in wired networks. Moreover, hosts are battery-operated, so they may shut because of energy shortage.

In this paper we give attention exactly to this problem, in the case of radio networks and with the objective of minimizing the total increase of energy consumption. Moreover, we will assume that all established links are *bidirectional* or *symmetric*. In the symmetric link model, if a node s is assigned to send messages to a node t, then t must directly send messages to s, too. In practice, symmetric links are desirable because they greatly simplify routing protocols.

Organization of the paper. The paper is organized as follows: in Section 2 we describe the general problem and give an overview of the current state

art; in Section 3 we prove inapproximability results for different variants of our problem; in Section 4 we study the relations between our problem and the corresponding problem in wired networks; finally, in Section 5 we describe approximation algorithms for the general problem in the case of either a link or a node failure.

2 General Problem Statement and Related Work

General problem statement. In the general *Range Augmentation Problem* (RAP) we are given a 4-tuple $\langle V, c, p_0, \pi \rangle$ where:

- V is the set of n stations (or nodes);
- $c : V \times V \mapsto \mathbb{R}^+ \cup \{+\infty\}$ is the *transmission power cost function*. With a little abuse of notation, we will denote by uv the ordered pair of stations (u, v)[1];
- $p_0 : V \mapsto \mathbb{R}^+$ is the *initial power assignment*;
- π is a network connectivity predicate;

and we are asked for a *minimum total power augmentation assignment* p, i.e., a function $p : V \mapsto \mathbb{R}^+$ that minimizes $\sum_{v \in V} p(v)$ and such that $\pi(p_f)$ is satisfied, where $p_f(v) = p_0(v) + p(v)$ for each station v.

Therefore a minimum total power augmentation assignment p specifies how the nodes must increase their current transmission power in order to satisfy property π, and in such a way that the total increase of energy consumption is minimized.

Problem classes. The above RAP definition is powerful and contains a wide range of problems. First note that if p_0 is the constant zero function, then the problem of building a minimum total power network from scratch satisfying property π is a special case of RAP. Moreover, predicate π may range over a very large set of properties. A first classification of such properties may be the following:

- π is a property related to the *asymmetric-link network induced from* p_f. Such a network can be modelled through a communication digraph $D_{p_f} = (V, E)$, where $E = \{uv | u, v \in V \wedge p_f(u) \geq c(uv)\}$;
- π is a property related to the *symmetric-link network induced from* p_f. Even in this case, such a network can be modelled through a communication graph $G_{p_f} = (V, E)$, where E contains the unordered pair (u, v) iff $p_f(u) \geq c(uv) \wedge p_f(v) \geq c(vu)$.

Another important parameter of RAP is the transmission power cost function c. First of all, the function c may be *symmetric*, i.e., $c(uv) = c(vu)$ for every

[1] Whenever the transmission power cost function of a pair uv is not defined, it is assumed to be $+\infty$.

pair of stations u, v[2]. Moreover, in any realistic setting (see [21]), we may have a distance function d[3] and a transmission-power gradient $\alpha \geq 1$.Henceforth, unless stated otherwise, we denote by c_s a symmetric transmission power cost function and whenever it is a distance we will denote it by d.

As a direct consequence of the above discussion, we need to classify RAP. We argue that the important parameters are c, p_0 and π. Therefore, we will denote by RAP$[c, p_0, \pi]$ a class of RAP, and for simplifying the notation, we will denote the property π throughout a mnemonic name. For instance, the connectivity RAP with asymmetric power cost function and the initial power assignment to be equal to 0 will be denoted by RAP$[c, 0, \mathrm{C}]$, where 0 is the constant function 0[4], and C stays for connectivity. By RAP$[c_s, 0, \mathrm{SC}]$ we will denote the strong connectivity RAP with symmetric power cost function and with the initial power assignment to be equal to 0.

It is worth noticing that any "negative" result for RAP$[c_s, p_0, \pi]$ holds for its asymmetric power cost function version, i.e., RAP$[c, p_0, \pi]$, as it is a special case.

Related work. The objective of minimizing the total power has been addressed under many specific properties π. Among them, we remind the following:

- In [2] it is claimed that RAP$[c, 0, \mathrm{C}]$ can not be approximated within $(1 - o(1)) \ln n$, unless $\mathsf{NP} \subset \mathsf{DTIME}(n^{\mathcal{O}(\log \log n)})$. The same paper presents a factor $2+2\ln(n-1)$ approximation algorithm for such a problem. Its symmetric version is MAX-SNP-hard [8]. For this latter problem there is a polynomial time approximation scheme with a performance guarantee of $5/3 + \epsilon$ [1].
- In [2] it is also claimed that RAP$[c, 0, \mathrm{SC}]$ is as hard as the Set Cover Problem. The paper contains also a factor $3 + 2\ln(n-1)$ approximation algorithm for such a problem. Its symmetric version has been proved to be MAX-SNP and factor 2 approximation algorithms are known for it [5, 17].
- RAP$[c_s, 0, k\mathrm{E(V)C}]$, where kE(V)C stays for k-Edge (Vertex) Connectivity[5], is NP-hard when $k = 2$ [4]. These results hold even when the stations are located onto a plane, and the transmission power cost function is given by the Euclidean distance function. Approximation algorithms for both problems come out from a careful analysis of approximation algorithms for k-

[2] Assume that $+\infty$ can be compared.

[3] Function d is a *distance* if, for each pair of stations u, v:

- $d(uv) \geq 0$ (it is assumed that $d(ww) = 0$ for every station w);
- $d(uv) = d(vu)$;
- $d(uv) \leq d(uw) + d(wv)$, for every station w (*triangle inequality*).

[4] Henceforth, whenever p_0 will be denoted by a constant $\rho \in \mathbb{R}^+$, then we mean $p_0(v) = \rho$ for every station v.

[5] From now on, whenever we will refer to both k-edge and k-vertex connectivity problems, we will write k-connectivity or simply kC.

edge (vertex) connectivity problems for wired networks[6] when applied to radio networks (see [4, 13]). However, such algorithms do not perform as well as in the wired case, since the objective function to minimize is different. Călinescu and Wan [4] proved that the 2-approximation algorithm of Khuller and Ragavachari [15] for kEC, has a performance guarantee of $2k$ when applied to RAP$[c_s, 0, k\text{EC}]$. In the same paper it has been proved that the 2-approximation algorithm of Khuller and Vishkin [16] for 2VC is a factor 4 approximation algorithm for RAP$[c_s, 0, 2\text{VC}]$. In the paper of Hajiaghayi, Immorlica and Mirrokni [13], the authors proved that any factor λ approximation algorithm for kC is an algorithm for RAP$[c_s, 0, k\text{C}]$ with a performance guarantee of $2\lambda k$. Moreover, by a careful analysis of the factor k approximation algorithm of Kortsarz and Nutov [18] for kVC, they proved that such an algorithm has a performance guarantee of $8(k-1)$ for RAP$[c_s, 0, k\text{VC}]$ with a slight improvement for values of $k = 4, 5, 6, 7$.

For a survey of other RAP classes, such as *unicast*, *broadcast*, *multicast*, etc., see [3, 6, 7, 20, 22, 24].

In this paper we focus on RAP$[c, p_0, k\text{CA}]$, where kCA stays for k-(either edge or vertex) *Connectivity Augmentation*, when the initial range assignment p_0 induces a connected graph.

3 Inapproximability Results

In this section we prove some inapproximability results for RAP$[c_s, p_0, 2\text{CA}]$ through reductions from the *Set Cover Problem*.

In the *Smallest Set Cover Problem* (SCP) we are given a set $U = \{o_1, \ldots, o_m\}$ of m objects and a collection $\mathscr{S}$ of h subsets $S_1, \ldots, S_h$ of U. We are asked for the smallest cover C^*, i.e., the minimum cardinality collection $C^* \subseteq \mathscr{S}$ such that $\bigcup_{S \in C^*} S = U$. In the weighted version of the SCP (WSCP) at each set S is associated a real positive value $w(S)$ (called *weight*) and the objective function is to minimize the total weight of the cover $w(C^*) = \sum_{S \in C^*} w(S)$.

Both problems are not-approximable within a factor better than $(1-o(1)) \ln m$, unless $\mathsf{NP} \subset \mathsf{DTIME}(m^{\mathcal{O}(\log \log m)})$ [10]. Moreover, there exists a simple factor $\mathcal{H}_m$ approximation greedy algorithm (see [23]), where $\mathcal{H}_m = \sum_{i=1}^{m} 1/i$ is the *m-th harmonic number*. We remind that, for every $m \geq 1$, $\ln(1+m) \leq \mathcal{H}_m \leq 1 + \ln m$.

Let P be an optimization problem and I an instance for P. We denote by $\mathtt{opt}_P(I)$ an optimal solution for I when the problem is P. If the instance will be clear from the context, we omit I from notation. With a little abuse of notation, we will denote by $\mathtt{opt}_P$ the measure of $\mathtt{opt}_P$ whenever it will not create confusion.

[6] In the *k-edge (vertex) connectivity problem* ($kE(V)C$) we are given a weighted undirected graph $G = (V, E)$, with weight function $w : E \mapsto \mathbb{R}^+$ and we are asked for finding a subset of edges E^* of E, such that the graph $G^* = (V, E^*)$ is k-edge (vertex) connected and $\sum_{e \in E^*} w(e)$ is minimized.

Theorem 1. RAP$[d, p_0, 2\text{CA}]$ *is not-approximable within a factor better than* $(1-o(1))\ln n$*, unless* NP $\subset$ DTIME$(n^{\mathcal{O}(\log\log n)})$*, even when the distance function* d *is restricted to three power levels.*

Proof. Let U and $\mathscr{S}$ be an instance of SCP. Let $V = \{r\} \cup V_U \cup V_{\mathscr{S}}$, where $V_U = \{v_i \mid o_i \in U\}$ and $V_{\mathscr{S}} = \{u_i \mid S_i \in \mathscr{S}\}$. Let $d(u_i, u_j) = 2$, for different $u_i, u_j \in V_{\mathscr{S}}$. Let $d(r, v_i) = 3, \forall v_i \in V_U$, $d(r, u_i) = 2, \forall u_i \in V_{\mathscr{S}}$, and let $d(v_i, v_j) = 5$ for different $v_i, v_j \in V_U$. If $o_i \in S_j$ then $d(v_i, u_j) = 3$, otherwise $d(v_i, u_j) = 5$. Let $p_0(r) = 3, p_0(v_i) = 3$, for each $v_i \in V_U$, and let $p_0(u_i) = 2$ for each $u_i \in V_{\mathscr{S}}$ be the initial power assignment (see Figure 1). It is easy to prove that d is a distance function and that G_{p_0} is connected.

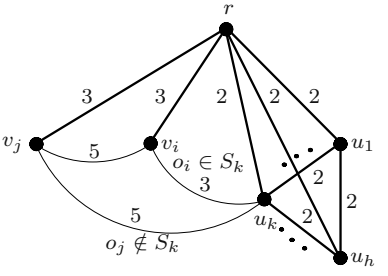


Fig. 1. The generic reduction from SCP instances into RAP$[d, p_0, 2\text{CA}]$ instances as explained in Theorem 1. Bold edges represent the connected graph induced from the initial power assignment p_0. Note that the edges satisfy the triangle inequality

Now, let C be a solution for the SCP. We transform C into a solution for RAP$[d, p_0, 2\text{CA}]$ in the following way: for each $v_i \in V_U$ we set $p(v_i) = 0$, while for each $u_j \in V_{\mathscr{S}}$, if $S_j \in C$ then $p(u_j) = 1$, otherwise $p(u_j) = 0$. Note that the total power augmentation assignment p equals the cardinality of C.

Now we prove that any optimal solution p^* for RAP$[d, p_0, 2\text{CA}]$ is a transformation of a solution for SCP. Note that it suffices to prove that $G_{p_0+p^*}$ contains no edge having cost 5. Indeed, if this is true, then $p^*(v_i) = 0$, $\forall v_i \in V_U$, while $p^*(u_j)$, with $u_j \in V_{\mathscr{S}}$, may be either 0 or 1. Moreover, as $\forall v_i \in V_U$, (r, v_i) is a *bridge*[7] in G_{p_0}, then $G_{p_0+p^*}$ must contain at least one edge *covering* (r, v_i), and so at least one node $u_j \in V_{\mathscr{S}}$ such that $o_i \in S_j$ must increase its power by one (i.e., $p^*(u_j) = 1$).

It remains to prove that any optimal solution p^* for RAP$[d, p_0, 2\text{CA}]$ is such that $G_{p_0+p^*}$ does not contain any edge having cost 5. If not so, then note that such edges must be incident to some $v_i \in V_U$. For each of such v_i's take a vertex $u_j \in V_{\mathscr{S}}$ such that $c(v_i, u_j) = 3$ (note that such a vertex always exists as object o_i is covered by some set in $\mathscr{S}$), and set $p(v_i) = 0$ and $p(u_j) = 1$ (note that

[7] A *bridge* of a graph $G = (V, E)$ is an edge whose removal from G disconnects it. Bridge $b \in E$ is *covered* by a set of edges F, with $F \cap E = \emptyset$, if F contains an edge whose addition to G makes b to be not a bridge.

$p^*(v_i) = 2$). For the remaining nodes $v \in V$ set $p(v) = p^*(v)$. Now it is easy to see that p is a better solution for RAP$[d, p_0, 2\text{CA}]$ than p^*.

Since the transformation function restricted to the set of optimal solutions is a bijection and preserves the costs[8], then the theorem follows from the inapproximability results for SCP [10]. □

One may think that the problem becomes "easier" if the initial power assignment induces a tree. The following theorem shows that this is not the case:

Theorem 2. RAP$[d, p_0, 2\text{CA}]$ *is not-approximable within a factor better than* $(1 - o(1)) \ln n$*, unless* NP $\subset$ DTIME$(n^{\mathcal{O}(\log \log n)})$*, even when* G_{p_0} *is a tree.*

Proof. Let U and $\mathscr{S}$ be an instance of SCP and let V, p_0 and d be defined as in Theorem 1 except for different $u_i, u_j \in V_{\mathscr{S}}$, for which $d(u_i, u_j) = 2 + \delta$.

By proceeding as in Theorem 1, it is not hard to prove that, for small values of δ, the transformation function restricted to the set of optimal solutions is a bijection. Since the transformation preserves costs, except for a linear additive function $f(\delta)$ such that $h\delta \geq f(\delta) \xrightarrow{\delta \mapsto 0} 0$[9], then the theorem follows from the inapproximability results for SCP [10]. □

An interesting variant of RAP$[c, p_0, 2\text{CA}]$ is when we treat with the so called *uniform* instances, i.e., instances for which the initial power assignment is the same for each node. Even if in the next section we prove that this simplification let the problem become "easier", it remains "intractable".

Theorem 3. RAP$[d, \rho \in \mathbb{R}^+, 2\text{CA}]$ *is* APX*-hard, unless* NP $\subset$ DTIME$(n^{\mathcal{O}(\log \log n)})$.

Proof. Let U and $\mathscr{S}$ be an instance of SCP and let V be defined as in Theorem 1. Let $d(u_i, u_j) = 2$, for different $u_i, u_j \in V_{\mathscr{S}}$. Let $d(r, w) = 2, \forall w \in V \setminus \{r\}$, and let $d(v_i, v_j) = 4$ for different $v_i, v_j \in V_U$. If $o_i \in S_j$ then $d(v_i, u_j) = 3$, otherwise $d(v_i, u_j) = 4$. Let $p_0(v) = 2$ for every station v.

By proceeding as in Theorem 1, it is not hard to prove that the transformation function restricted to the set of optimal solutions for both instances is a bijection. Let $\texttt{opt}_{\text{RAP}}$ denote an optimum for RAP$[d, \rho, 2\text{CA}]$. By construction, we have that $\texttt{opt}_{\text{RAP}} = m + \texttt{opt}_{\text{SCP}}$, where $|U| = m$. Since SCP is not-approximable within $(1 - o(1)) \ln m$, unless NP $\subset$ DTIME$(m^{\mathcal{O}(\log \log m)})$ [10], then under the same hypothesis, RAP$[d, \rho, 2\text{CA}]$ is not-approximable within a factor better than $\frac{m + (1 - o(1))(\ln n)\texttt{opt}_{\text{SCP}}}{m + \texttt{opt}_{\text{SCP}}}$. See Corollary 1 to complete the proof. □

Theorem 4. RAP$[d, \rho \in \mathbb{R}^+, 2\text{CA}]$ *is* APX*-hard, unless* NP $\subset$ DTIME$(n^{\mathcal{O}(\log \log n)})$*, even when the initial power assignment* p_0 *induces a tree.*

Proof. Combines the transformations described in Theorems 2 and 3 and the proof of Theorem 3. □

[8] We can implicitly assume that each set $S \in \mathscr{S}$ has weight 1.

[9] Remember that $|\mathscr{S}| = h$.

Next theorem captures an inapproximability result for the RAP$[c_s, p_0, 2\text{CA}]$, whenever we have an existing communication network of nodes located onto a Euclidean plane and we want to increase its reliability, but not all the pairs of nodes are allowed to communicate with each other.For instance, such a situation may happen when there is a set A of agents, each owning some base nodes, and there is a set R of clients, with $R \cap A = \emptyset$. Assume each client $r \in R$ is a subscriber of a subset of agents $A_r \subseteq A$, so it is allowed to communicate only with base nodes owned by agents in A_r. Then, in order to guarantee a better service, such as improving the network reliability, the agents might commonly decide to augment their base nodes transmission power.

Theorem 5. RAP$[c_s, p_0, 2\text{CA}]$ *is not-approximable within a factor better than* $(1 - o(1)) \ln n$*, unless* NP $\subset$ DTIME$(n^{\mathcal{O}(\log \log n)})$*, even when the stations are located onto a 2-dimensional plane and* c_s *is the Euclidean distance function, whenever it is different from* $+\infty$*.*

Proof. Let U and $\mathscr{S}$ be an instance of SCP. Let $V = \{r\} \cup V_U \cup V_{\mathscr{S}}$ where $V_U = \{v_i \mid o_i \in U\}$ and $V_{\mathscr{S}} = \{u_i \mid S_i \in \mathscr{S}\}$. Let r be the center of two concentric circumferences with radius 1 and $\delta < 1$, respectively. Let u_i's be located all over the circumference with radius δ, while v_i's are located all over the other circumference. Assume that v_i's can not directly communicate with each other. Moreover v_j's and u_i's can directly communicate with each other iff $o_j \in S_i$. Let $p_0(r) = p_0(v_i) = 1$ and let $p_0(u_i) = 2\delta$ (see Figure 2).

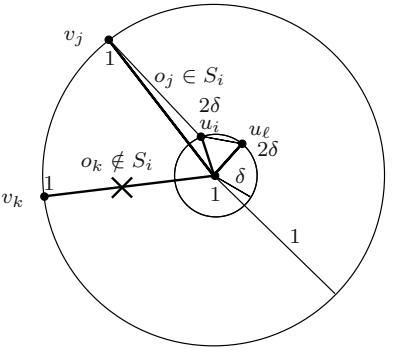


Fig. 2. The generic transformation from SCP instances into particular RAP$[c_s, p_0, 2\text{CA}]$ instances as explained in Theorem 5. For each unordered pair of stations (u, v), either $c_s(u, v) = +\infty$ or it is equal to the Euclidean distance between the two stations. Bold edges represent the connected graph induced from the initial power assignment p_0

By proceeding as in Theorem 1, it is not hard to prove that, for small values of δ, the transformation function restricted to the set of optimal solutions for both instances is bijective. Since the transformation preserves costs, except for a linear polynomial $P(\delta)$, that tends to zero for small values of δ, then the theorem follows from the inapproximability results for SCP [10]. □

Remark 1. The inapproximability result of Theorem 5 can be extended for the broadcast RAP. The proof is quite similar.

4 Relations with Wired Augmentation Problems

As the best actually known algorithms for RAP$[c_s, 0, k\text{C}]$ are the same of the corresponding problem for wired networks (see [4, 13]), then for RAP$[c_s, p_0, k\text{CA}]$ we first analyze the relations with its wired corresponding problem.

In the k-(either edge or vertex) *Connectivity Augmentation Problem* (kCAP) we are given a weighted undirected graph $G = (V, E)$, with weight function $w : E \mapsto \mathbb{R}^+$, and a spanning connected subgraph H of G. We are asked for a set of edges $F \subseteq E$ whose addition to H makes it to be k-edge (vertex) connected and such that $w(F) = \sum_{e \in F} w(e)$ is minimized. Both problems have been proved to be NP-hard [9, 11] for $k = 2$ (for a survey on kCAP see [14]).

The following definitions and lemmas turn out to be useful for our purposes.

Definition 1. *Given a weighted digraph $D = (V, E)$, with non negative weight function c, the* power induced *from D to a host $v \in V$ (denoted by $p_f^D(v)$) is $\max_{vu \in E} c(vu)$, while the* total power assignment induced *from D (denoted by $p_f(D)$), is $\sum_{v \in V} p_f^D(v)$. The extension of these definitions for a weighted undirected graph G are straightforward*[10].

Definition 2. *Given a weighted digraph $D = (V, E)$, with non negative weight function c and an initial power assignment p_0, the* total additional power assignment induced *from D (denoted by $p(D)$), is $\sum_{v \in V} \left(p_f^D(v) - p_0(v) \right)$. The extension of this definition for a weighted undirected graph G is straightforward.*

Lemma 1. *[17] For any undirected graph $G = (V, E)$, $p_f(G) \leq 2c(G)$*[11]. □

Lemma 2. *[17] For any undirected forest $\mathcal{F}$, $c(\mathcal{F}) \leq p_f(\mathcal{F})$.* □

Lemma 3. *Let $\mathcal{F}$ be a feasible forest*[12] *for* RAP$[c, p_0, 2\text{CA}]$ *that minimizes $p(\mathcal{F})$. Then $p(\mathcal{F})$ is an optimum for* RAP$[c, p_0, 2\text{CA}]$.

Proof. For the sake of contradiction let us assume that $p(\mathcal{F})$ is not an optimal solution, that is, there exists a feasible range assignment p^*, such that

[10] An undirected edge (u, v) with weight w can be viewed as two directed edges uv and vu both having weight w.

[11] $c(G) = \sum_{e \in E} c(e)$.

[12] The union between G_{p_0} and $\mathcal{F}$ generates a 2-edge (vertex) connected graph.

$\sum_{v \in V} p^*(v) < p(\mathcal{F})$ and $G' = G_{p^*+p_0} - G_{p_0}$ is not a forest[13]. We prove that we can remove edges from G' until there is no cycle left and the resulting graph is still a cover for T, where T is the *bridge graph*[14] (resp., *cut-node graph*[15]) of G_{p_0}. Let $(v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, v_1)$ be a cycle in G' and consider the subtree T' of T covered by this cycle. If we prove that each edge (resp., vertex) of T' is covered by at least 2 edges of the cycle, then we can remove any of such edges and the remaining edges still form a cover for T'.

For the sake of contradiction, let us assume that there is an edge (resp., vertex) e of T' covered by only one cycle edge, say (v_j, v_{j+1}). The removal of e and (v_j, v_{j+1}) from T' splits T' into two subtrees T_1, T_2, such that v_j is in T_1 and v_{j+1} is in T_2. Since a cycle is a 2-edge connected graph, then there is another cycle edge, say (v_i, v_{i+1}), with $i \neq j$, that covers e. So we can remove any edge of the cycle obtaining a cover for T', and the claim follows. □

Remark 2. The results in Lemma 3 can be easily extended for 2CAP.

At this point we have sufficient material for relating 2CAP with RAP$[c, p_0, 2\text{CA}]$. Unfortunately, the relation between the two problems is "rather slack" even if we restrict ourself to the case of symmetric cost function.

Theorem 6. *Any λ-approximation algorithm $\mathcal{A}$ for* 2CAP *is a factor $n\lambda$ approximation algorithm for* RAP$[c, p_0, 2\text{CA}]$.

Proof. Build a complete edge-weighted (weight function w) undirected graph G on V as follows. For every pair of nodes u, v, let $w(u, v) = w_u(v) + w_v(u)$, where $w_u(v) = \max\{0, c(uv) - p_0(u)\}$ and $w_v(u) = \max\{0, c(vu) - p_0(v)\}$[16].

From Remark 2 and Lemma 3 we have that any minimal (w.r.t. edge removal) solution for 2CAP and RAP$[c, p_0, 2\text{CA}]$ is a forest. So let $\mathcal{F}' = (V, E')$ denote the forest solution computed by the algorithm $\mathcal{A}$ when applied to the instance G, w, G_{p_0}, while $\mathcal{F}_{2\text{CAP}}$ denotes a forest that is an optimal solution for it. Moreover, let $\mathcal{F}^* = (V, E^*)$ denote a forest that is an optimal solution for

[13] G' is the graph obtained by removing the edges in G_{p_0} from $G_{p^*+p_0}$.

[14] The *bridge graph* of a graph G is a tree T obtained by repeatedly coalescing into a node non-bridge edges of G until there are only bridges.

[15] A *cut-node* of a graph $G = (V, E)$ is a node whose removal from G disconnects it. A cut-node v is *covered* by a set of edges $F \cap E = \emptyset$, if F contains an edge whose addition to G makes v to be not a cut-node.The *cut-node graph* of a graph G is a tree T obtained by:

- coalescing each *2-vertex connected components* C (i.e., subset of vertices in G which induces a 2-vertex connected graph) into a node u_C;
- adding the set of cut-nodes;
- connecting a cut-node v to a vertex u_C representing a 2-vertex connected component, iff $v \in C$.

[16] Assume that $+\infty \pm \rho = +\infty$ for any $\rho \in \mathbb{R}^+$.

RAP$[c, p_0, 2\text{CA}]$. Then as a consequence of Lemma 1 (we remind that the cost function is c) we have that:

$$p(\mathcal{F}') = \sum_{v \in V} \left(p_f^{\mathcal{F}'}(v) - p_0(v) \right) = \sum_{v \in V} \max_{u | vu \in E'} \{w_v(u)\} \leq w(\mathcal{F}')$$
$$\leq \lambda w(\mathcal{F}_{2\text{CAP}}) \leq \lambda w(\mathcal{F}^*) = \lambda \sum_{v \in V} \sum_{u | vu \in E^*} w_v(u) \leq n\lambda p(\mathcal{F}^*).$$

□

In Figure 3 we give an example showing that an optimum for 2CAP is a $\Omega(n)$-approximation for RAP$[c_s, p_0, 2\text{CA}]$ even if we decide to apply the transformation described in Theorem 6 or not. By contrast to the previous result, a better result holds for the case in which the initial range assignment is identical for all stations (we remind that 2CAP is approximable within a factor 2 [14]).

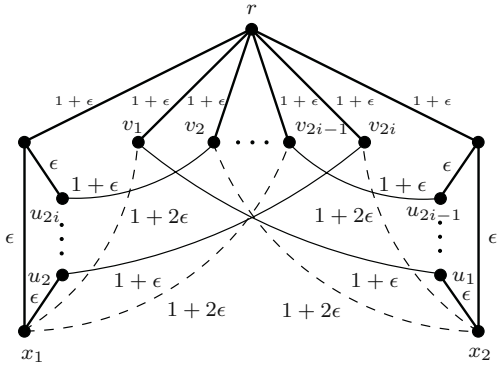


Fig. 3. An example for the result in Theorem 6. Bold edges represent the initial connected graph. The initial power assignment is $1 + \epsilon$ for r and v_i's, while it is ϵ for x_1, x_2 and u_i's. The optimum for 2CAP is given by the solid thin edges, which induces a total power augmentation assignment of $2i = \Omega(n)$. The optimum power assignment is given by assigning power $1 + 2\epsilon$ to x_1, x_2 for a total power augmentation of $2 + 2\epsilon$. Even if we apply the transformation described in Theorem 6, we have that the solid thin edges have a weight w equal to 1 while the dashed one a weight equal to $1 + 2\epsilon$ and the gap between the power induced from an optimum for 2CAP and an optimum power assignment still remains $\Omega(n)$

Corollary 1. *Any λ-approximation algorithm $\mathcal{A}$ for* 2CAP *is a factor 2λ approximation algorithm for* RAP$[c_s, \rho \in \mathbb{R}^+, 2\text{CA}]$.

Proof. The proof is identical to the one described in Theorem 6 except for the last inequality of the equation where the term n can be replaced by 2, as a direct consequence of Lemma 2 along with the definition of w (note that as c_s is symmetric and the initial power assignment is uniform, then $w_u(v) = w_v(u)$, for every pair of stations u, v). □

5 Approximation Algorithms

In this section, we describe approximation algorithms for $\textsc{Rap}[c, p_0, 2\text{CA}]$. We start by treating $\textsc{Rap}[c, p_0, 2\text{ECA}]$, namely the edge-connectivity case. We first build a weighted digraph $D = (V, E)$, with weight function $w : E \mapsto \mathbb{R}^+ \cup \{+\infty\}$, representing the *residual network* w.r.t. the initial range assignment p_0. More formally, for each pair of nodes $u, v \in V$, $w(uv) = +\infty$ iff (u, v) is either an edge in G_{p_0} or $c(u, v) = +\infty$, otherwise $w(uv) = \max\{0, c(uv) - p_0(u)\}$. We use the ideas in [2] for approximating $\textsc{Rap}[c, p_0, 2\text{ECA}]$.

Definition 3. *A* star *S is a tree consisting of one* center *and several* leaves *adjacent to the center. The* weight *of the star, denoted by $w(S)$ is given by the sum of weights of the arcs from the leaves to the center plus the maximum weight chosen among the arcs going from the center to the leaves.*

Definition 4. *A star S is* bridge cost-effectiveness *w.r.t. graph H, if it minimizes $w(S)/|B(S|H)|$, where $B(S|H)$ denotes the set of bridges of H covered by S.*

Next we describe an algorithm for $\textsc{Rap}[c, p_0, 2\text{ECA}]$.

Input: A weighted digraph $D = (V, E)$, with weight function w and a graph G_{p_0}.
Output: A power assignment augmentation p for $\textsc{Rap}[c, p_0, 2\text{ECA}]$.
begin
  $H \leftarrow G_{p_0}, H' \leftarrow (V, \emptyset)$
  while H is not 2-edge connected **do**
    $S' \leftarrow \lambda$-apx of a bridge cost-effectiveness star S w.r.t. H
    $price(b) = w(S')/|B(S'|H)|, \forall b \in B(S'|H)$
    $H \leftarrow H \cup S'$ {*Here S' is considered as undirected*}
    $H' \leftarrow H' \cup S'$
  end while
  for all $v \in V$, $p(v) = \max_{u|(v,u)\in E(H)} w(vu)$
  output p
end.

Number the bridges of G_{p_0} in the order in which they were covered by the algorithm. Let $b_1, b_2, \ldots, b_m$, with $m < n$, be this numbering, assuming G_{p_0} has m bridges. In the following lemma, we let `opt` denote an optimum solution for $\textsc{Rap}[c, p_0, 2\text{ECA}]$. The proofs of the following lemma and theorem combine the ideas in [2] together with the performance analysis of the greedy algorithm for Wscp [23].

Lemma 4. *For each $k \in 1, \ldots, m$, $price(b_k) \leq 2\lambda\texttt{opt}/(m - k + 1)$, where λ is the approximation factor for a bridge cost-effectiveness star.*

Proof. In any iteration, the remaining stars of the optimal solution cover the uncovered bridges with a total cost of at most $2\texttt{opt}$. This is true because from Lemma 3 we have that the optimum solution is a forest $\mathcal{F} = \cup_{i=1}^{\ell} \mathcal{T}_i$, where

$\ell < m$. If we root each $\mathcal{T}_i$ arbitrarily and for every vertex v, we define the star S_v consisting of v and all its children, then as proved in [2] for a tree, we have

$$\sum_{v \in V} w(S_v) \leq 2\texttt{opt}$$

as a vertex u contributes to $\sum_{v \in V} w(S_v)$ for at most two stars and its contribution in each star is at most the power induced from $\mathcal{F}$ to u. Therefore, among the remaining stars, there must be one having bridge cost-effectiveness of at most $2\texttt{opt}/b$, assuming there are b bridges uncovered. In the iteration in which bridge b_k was covered, b was at least $m - k + 1$. For the λ-approximated greedy choice, it follows that

$$price(b_k) \leq \frac{2\lambda\texttt{opt}}{b} \leq \frac{2\lambda\texttt{opt}}{m-k+1}.$$

□

Now we can state the following:

Theorem 7. *The greedy algorithm is a factor $2\lambda\mathcal{H}_n$ approximation algorithm for* Rap$[c, p_0, 2\text{ECA}]$*, where λ is the factor approximation for a bridge cost-effectiveness star.*

Proof. Since the cost of each star picked is distributed among the new bridges covered, the total power induced from the stars picked is less or equal than $\sum_{i=1}^{m} price(b_i)$. By the previous lemma, this is at most

$$2\lambda\left(\frac{1}{m} + \frac{1}{m-1} + \cdots + \frac{1}{2} + 1\right)\texttt{opt} = 2\lambda\mathcal{H}_m\texttt{opt} \leq 2\lambda\mathcal{H}_n\texttt{opt}.$$

□

5.1 Approximating the Cost-Effectiveness Star

Next we describe how to find a $\mathcal{H}_n$-approximation for the bridge cost-effectiveness star. We use the ideas in [2].

Definition 5. *A* powered star *is a star with a fixed power $p(v)$ associated with the center v, and all the arcs from the center to the leaves have weight at most $p(v)$. The* weight *of the powered star S' with center v and power $p(v)$, denoted by $w(S')$, is equal to $p(v)$ plus the sum of the weights of the arcs from the leaves to the center.*

One can prove that if S' is a powered star minimizing $w(S')/|B(S'|H)|$, then by ignoring the fixed power of its center, the same star is bridge cost-effectiveness w.r.t. H. Therefore, we describe an algorithm for approximating the bridge cost-effectiveness powered star. So, for each possible center $v \in V$ and each other node $v' \in V \setminus \{v\}$, let $p(v) = w(vv')$ be the fixed power associated with the center. We build the following instance of *Partial Weighted Set Cover Problem* (Pwscp). For

each node u with which v can communicate directly by setting its transmitting power to $p(v)$, define the set S_u as the set of bridges of H covered by (u, v), and assign it a weight $w(S_u) = w(uv)$. Let U be the set of bridges of H covered by some set S_u. The objective is to find a partial collection of sets $\mathcal{C}^*$ that minimizes $(p(v)+\sum_{S\in\mathcal{C}^*} w(S))/|\cup_{S\in\mathcal{C}^*} S|$. It is easy to prove that this transformation maps optimal solutions for the problem of finding the cost-effectiveness powered star into optimal solutions for PWSCP. Moreover the transformation preserves the weights.

For solving our problem, we use a modified version of the well-known factor $\mathcal{H}_m$ approximation greedy algorithm for WSCP [23]. The greedy algorithm for WSCP starts with a solution $C = \emptyset$, and until C is not a cover for U, it picks a *cost-effectiveness set* w.r.t. the uncovered objects, i.e., the set minimizing the ratio between its cost and the number of new objects covered, and adds it to C. Our picking strategy is slightly different; we pick a cost-effectiveness set and adds it to C until the objective function improves. The following statement holds:

Lemma 5. *There exists a factor $\mathcal{H}_m$ approximation greedy algorithm for* PWSCP.

Proof. Assume that $\mathtt{opt}_{\text{PWSCP}}$ covers $k \leq m$ objects. Consider the instance of WSCP obtained from PWSCP by simply ignoring $p(v)$. Let $\mathtt{opt}^k_{\text{WSCP}}$ denote the optimum collection of sets covering at least k objects. As a consequence of the performance analysis of the greedy algorithm for WSCP [23], and because of the non negative value $p(v)$, we have that our solution has a total cost less or equal than $\frac{p(v)+\mathcal{H}_k \mathtt{opt}^k_{\text{WSCP}}}{k} \leq \mathcal{H}_m \mathtt{opt}_{\text{PWSCP}}$. □

As a direct consequence of Theorem 7 and Lemma 5, we have:

Corollary 2. RAP$[c, p_0, 2\text{ECA}]$ *can be approximated within a factor of* $2\mathcal{H}^2_n$. □

Now we extend the RAP$[c, p_0, 2\text{ECA}]$ algorithm for the vertex-connectivity case. Let $C(S|H)$ be the set of cut-vertices of H covered by S. We say that a star S is *cut-node cost-effectiveness* w.r.t. graph H, if it minimizes $w(S)/|C(S|H)|$. Now we can prove that:

Theorem 8. *There exists a factor $2\lambda\mathcal{H}_n$ approximation greedy algorithm for* RAP$[c, p_0, 2\text{VCA}]$*, where λ is the approximation factor for a cut-node cost effectiveness star.*

Proof. The algorithm is similar to the one described for RAP$[c, p_0, 2\text{ECA}]$. The picking strategy is the following: add a λ-approximation of the cut-node cost-effectiveness star to H until it becomes 2-vertex connected. Note that an analogous of Lemma 4 still holds if λ is the approximation factor for the cut-node cost-effectiveness star. □

Corollary 3. RAP$[c, p_0, 2\text{VCA}]$ *can be approximated within a factor of* $2\mathcal{H}^2_n$.

Proof. Define an instance of PWSCP as done for RAP$[c, p_0, 2\text{ECA}]$. The only difference is that S_u is the set of cut-nodes covered by (u, v). Note that Lemma 5 still holds. □

References

1. E. Althaus, G. Călinescu, I.I. Mandoiu, S. Prasad, N. Tchervenski, and A. Zelikovsky, Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks, *Wireless Networks*, accepted for publication. Availble online at `http://www.mpi-sb.mpg.de/~althaus/winet.pdf`.
2. G. Călinescu, S. Kapoor, A. Olshevsky, and A. Zelikovsky, Network lifetime and power assignment in ad-hoc wireless networks, In *Proc. of 11th European Symp. on Algorithms (ESA'03)*, LNCS 2832, 114–126, 2003.
3. G. Călinescu, I.I. Măndoiu, and A. Zelikovsky, Symmetric connectivity with minimum power consumption in radio networks, In *Proc. of the 2nd IFIP Int. Conf. on Theoretical Computer Science*, Montreal, August 2002.
4. G. Călinescu and P.-J. Wan, Range assignment for high connectivity in wireless ad-hoc networks, In *Proc. of the 2nd International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-MWN'03)*, 235–246.
5. W. Chen and N. Huang, The strongly connecting problem on multihop packet radio networks, *IEEE Transaction on Communications*, **37** (1989), 293–295.
6. T. Chu and I. Nikolaidis, Energy efficient broadcast in mobile ad-hoc networks, In *Proc. of the 1st Int. Conf. on Ad-Hoc Networks and Wireless (ADHOC-NOW'02)*, Toronto, Canada, 20-21 September 2002, 177–190.
7. A.E.F. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca, On the complexity of computing minimum energy consumption broadcast subgraphs, In *18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'01)*, LNCS 2010, 121–131, 2001.
8. A.E.F. Clementi, P. Penna, and R. Silvestri, On the power assignment problem in radio networks, *Electronic Colloquium on Computational Complexity (ECCC'00).*
9. K.P. Eswaran and R.E. Tarjan, Augmentation problems, *SIAM Journal on Computing* **5(4)** *(1976), 653–665.*
10. U. Feige, A treshold of $\ln n$ for approximating set cover, *Journal of the ACM,* **45** *(1998), 634–652.*
11. G.N. Frederickson and J. Jájá, Approximation algorithms for several graph augmentation problems, *SIAM Journal on Computing* **10(2)** *(1981), 270–283.*
12. Z. Haas and S. Tabrizi, On some challenges and design choices in ad-hoc communications, In *Proc. of the IEEE Military Communication Conference (MILCOM'98).*
13. M.T. Hajiaghayi, N. Immorlica, and V.S. Mirrokni, Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks, *Proc. of the 9th Annual International Conference on Mobile Computing and Networking, (MOBICOM'03)*, 300–312.
14. S. Khuller, Approximation algorithms for finding highly connected subgraphs, in *Approximation Algorithms for NP-Hard Problems*, D.S. Hochbaum Ed., PWS Publishing Co., Boston, MA, 1996.
15. S. Khuller and B. Ragavachari, Improved approximation algorithms for uniform connectivity problems, *Journal of Algorithms,* **21** *(1996), 433–450.*
16. S. Khuller and U. Vishkin, Biconnectivity approximation and graph carvings, *Journal of the ACM,* **41(2)** *(1994), 214–235.*
17. L.M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc, Power consumption in packet radio networks, *Theoretical Computer Science,* **243** *(2000), 289–305.*
18. G. Kortsarz and Z. Nutov, Approximating node connectivity problems via set covers, *Algorithmica* **37**(2) (2003), 75–92.
19. G.S. Lauer, Packet radio routing, in *Routing in Communication Networks*, M. Streenstup Ed., Prentice-Hall, 1995.

20. W. Liang, Constructing minimum-energy broadcast trees in wireless ad hoc networks, In *Proc. 3rd ACM Int. Symp. on Mobile Ad Hoc Networking and Computing (MOBIHOC'02)*, 112–122.
21. T.S. Rappaport, *Wireless Communications: Principles and Practices*, Prentice Hall, 1996.
22. G. Rossi, *The Range Assignment Problem in Static Ad-Hoc Wireless Networks*, Università di Roma "Tor Vergata", Ph.D. Thesis, 2003.
23. V.V. Vazirani, *Approximation Algorithms*, Springer-Verlag, 2001.
24. P.-J. Wan, G. Călinescu, X.-Y. Li, and O. Frieder, Minimum energy broadcast routing in static ad hoc wireless networks, In *Proc. of the IEEE Conf. on Computer Communications (INFOCOM'01)*, **2**, 1162–1171.

On the Approximability of the $L(h,k)$-Labelling Problem on Bipartite Graphs* (Extended Abstract)

Tiziana Calamoneri[1] and Paola Vocca[2]

[1] Dipartimento di Informatica, Università degli Studi di Roma "La Sapienza", via Salaria 113, 00198 Roma, Italy
calamo@di.uniroma1.it
[2] Department of Mathematics, University of Lecce - Italy, via Provinciale Lecce-Arnesano, P.O. Box 193,73100 Lecce, Italy
paola.vocca@unile.it

Abstract. Given an undirected graph G, an $L(h,k)$-labelling of G assigns colors to vertices from the integer set $\{0,\ldots,\lambda_{h,k}\}$, such that any two vertices v_i and v_j receive colors $c(v_i)$ and $c(v_j)$ satisfying the following conditions: *i)* if v_i and v_j are adjacent then $|c(v_i)-c(v_j)|\geq h$; *ii)* if v_i and v_j are at distance two then $|c(v_i)-c(v_j)|\geq k$. The aim of the $L(h,k)$-labelling problem is to minimize $\lambda_{h,k}$. In this paper we study the approximability of the $L(h,k)$-labelling problem on bipartite graphs and extend the results to s-partite and general graphs. Indeed, the decision version of this problem is known to be NP-complete in general and, to our knowledge, there are no polynomial solutions, either exact or approximate, for bipartite graphs.

Here, we state some results concerning the approximability of the $L(h,k)$-labelling problem for bipartite graphs, exploiting a novel technique, consisting in computing approximate vertex- and edge-colorings of auxiliary graphs to deduce an $L(h,k)$-labelling for the input bipartite graph. We derive an approximation algorithm with performance ratio bounded by $\frac{4}{3}D^2$, where, D is equal to the minimum even value bounding the minimum of the maximum degrees of the two partitions.

One of the above coloring algorithms is in fact an approximating edge-coloring algorithm for hypergraphs of maximum dimension d, i.e. the maximum edge cardinality, with performance ratio d.

Furthermore, we consider a different approximation technique based on the reduction of the $L(h,k)$-labelling problem to the vertex-coloring of the square of a graph. Using this approach we derive an approximation algorithm with performance ratio bounded by $\min(h,2k)\sqrt{n}+o(k\sqrt{n})$, assuming $h\geq k$. Hence, the first technique is competitive when $D=O(n^{1/4})$.

These algorithms match with a result in [2] stating that $L(1,1)$-labelling n-vertex bipartite graphs is hard to approximate within $n^{1/2-\epsilon}$, for any $\epsilon>0$, unless NP=ZPP.

* Partially supported by the Italian Research Project PRIN 2003 "Optimization, simulation and complexity of the design and management of communication networks".

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 65–77, 2005.

We then extend the latter approximation strategy to s-partite graphs, obtaining a $(\min(h, sk)\sqrt{n} + o(sk\sqrt{n}))$-approximation ratio, and to general graphs deriving an $(h\sqrt{n} + o(h\sqrt{n}))$-approximation algorithm, assuming $h \geq k$.

Finally, we prove that the $L(h, k)$-labelling problem is not easier than coloring the square of a graph.

1 Introduction

The *frequency assignment problem (FAP)* in a wireless network is a widely studied problem (see [1, 15] for a survey). A wireless network consists of a set of radio transmitter/receiver stations distributed over a region. Communication takes place by a node broadcasting a signal over a fixed range (whose size is proportional to the power expended by the node's transmitter). Any receiver within the range of the transmitter can receive the signal. In this context, the frequency assignment task is to assign radio frequencies to transmitters at different locations without causing interference. This situation can be modelled by a graph G, whose nodes are the radio transmitters/receivers, and the adjacencies indicate possible communications and, hence, interferences. To avoid interference, two adjacent stations must receive far frequencies. Therefore, the problem is closely related to graph coloring, where colors represent possible frequencies.

Many graph coloring models have been proposed to represent the FAP problems depending on the specific features of the problem (i.e. handling the interference, the availability of frequencies etc.) [15]. Among all these models the most widely accepted for the interferences avoidance is the $L(h, k)$-*labelling*, introduced by Griggs and Yeh [12] in the special case $h = 2$ and $k = 1$. The $L(h, k)$-labelling problem is a coloring problem with some constraints araising from practical reasons: a radio station and its neighbors must have far frequencies, at least h apart, so their signals will not interfere (*direct collision*); furthermore, a radio station must have a signal of frequency different at least k from the radio stations adjacent to its neighbors (*hidden collision*). The nature of the environment and the geographical distance are the major factors determining parameters h and k, and usually, $h \geq k$ holds.

It has been proven that the decision version of the $L(h, k)$-labelling problem is NP-complete even for $h \in \{1, 2\}$ and $k = 1$ [14, 13, 12]. Therefore, the problem has been widely studied for many specific classes of graphs. For some classes of graphs the problem has been proved to be polynomially solvable and for other classes to be approximable within a constant factor (see [7] for a survey on the $L(h, k)$-labelling, and, for instance, [6, 8, 10, 11, 18] for some specific results).

In this paper we focus on the approximability of the $L(h, k)$-labelling problem on bipartite graphs, for any constant $h \geq 2$ and $k \geq 1$. It is easy to see that when $h = k = 1$, the $L(h, k)$-labelling problem on a graph G is equivalent to

vertex-coloring G^2, where G^2 is the square graph of G, and a wide literature is known in this case (e.g., see [3, 14]).

Concerning **bipartite graphs**, previous best known results deal only with $h = 2$ and $k = 1$ [5], where the authors prove that the decision version of the $L(2,1)$-labelling problem is NP-complete also for planar bounded degree ($\Delta = 7$) bipartite graphs and they present an infinite class of bipartite graphs requiring at least $\frac{\Delta^2}{4}$ colors, where Δ is the maximum degree. We improve the lower bound by a constant factor, and we present two approximation algorithms whose approximation ratios depend on the degree and on the dimension of the vertex set, respectively. In particular the first algorithm exploits a novel technique, consisting in computing the approximate vertex- and edge-colorings of auxiliary graphs to deduce an $L(h,k)$-labelling for the input bipartite graph. The second approximation technique is based on the reduction of the $L(h,k)$-labelling problem to the vertex-coloring of the square graph. This strategy has the advantage to be extendable to s-partite and general graphs.

The obtained results are listed below:

- We improve the lower bound by a constant factor of $\frac{1}{4}$.
- For n-vertex general bipartite graphs we derive two approximation algorithms with performance ratio:
  - $\frac{4}{3}D^2$, where D is equal to the minimum even value bounding the minimum of the maximum degrees of the two partitions; this result is improved to $\frac{9}{2}$ if one of the two partitions has regular degree 2;
  - $\min(h, 2k)\sqrt{n} + o(k\sqrt{n})$, assuming $h \geq k$.

  Hence, the first technique is competitive when $D = O(n^{1/4})$.

These algorithms match with a result in [2] stating that $L(1,1)$-labelling n-vertex bipartite graphs is hard to approximate within $n^{1/2-\epsilon}$, for any $\epsilon > 0$, unless NP=ZPP.

The technique used to derive the approximation algorithm for general bipartite graphs straightforwardly derives from an edge-coloring approximation algorithm for hypergraphs. More precisely, given an n-vertex hypergraph $\mathcal{H}$ with maximum edge-cardinality d, we describe a d–approximation algorithm for the edge-coloring. To the knowledge of the authors, the best known previous results concern d–uniform hypergraphs, and the approximation ratio is a function of the vertex degree instead of the edge's dimension [4, 16].

Additionally, we extend the last result to **s-partite graphs**, obtaining a $(\min(h, sk)\sqrt{n} + o(sk\sqrt{n}))$-approximation ratio.

Finally, for what concerns **general graphs**:

- We present a $(h\sqrt{n} + o(h\sqrt{n}))$-approximation algorithm, assuming $h \geq k$;
- We prove that the $L(h,k)$-labelling problem is not easier than coloring the square of a graph.

Note that no results are known about the approximability of the $L(h,k)$-labelling problem on general graphs, while for the coloring of square graphs an $O(\sqrt{n})$-approximation algorithm exists [14]. In this context our result strongly relates the approximability of these two problems.

2 Definitions and Preliminary Results

In this section, we recall some basic concepts and known results, and introduce some definitions useful for the rest of the paper.

Definition 1. *An $L(h,k)$-labelling of a graph $G=(V,E)$ is a function f from V to the set of all nonnegative integers such that*
1. $|f(x)-f(y)| \geq h$ if x and y are at distance 1 in G;
2. $|f(x)-f(y)| \geq k$ if x and y are at distance 2 in G;
for some fixed integer values $h,k \geq 1$.

The *span* of an $L(h,k)$-labelling of a graph G is the difference between the maximum and minimum value of f. It is not restrictive to assume that the minimum value is 0, so the span coincides with the maximum value of f.

The *$L(h,k)$-number* of G, denoted by $\lambda^*_{h,k}(G)$ (or simply $\lambda^*(G)$, when the values of h and k are clear from the context), is the minimum span, over all $L(h,k)$-labellings of G. The task of the $L(h,k)$-labelling problem is to determine $\lambda^*(G)$. Although not necessary for our reasonings, we assume $h \geq k$, as suggested by real applications.

Let $G=(V,E)$ be a (multi)graph. In the following we denote by $\Delta(G)$ the maximum degree of G. Consider an optimal vertex-coloring of G and an optimal edge-coloring of G, let $\chi^*(G)$ and $\chi'^*(G)$ be its *chromatic number* and *chromatic index*, respectively, and let $\chi(G)$ and $\chi'(G)$ denote the number of colors used by an approximation algorithm for coloring vertices and edges of graph G, respectively.

Here, we recall some results relating these quantities.

Theorem 1. *[19, 17] The chromatic index of any graph G of maximum degree $\Delta(G)$ satisfies $\Delta(G) \leq \chi'^*(G) \leq \Delta(G)+1$. If G is a multigraph then $\chi'^*(G) \leq \frac{3}{2}\Delta(G)$.*

Since the proof is constructive, we have the following result:

Corollary 1. *There is an algorithm for coloring the edges of any graph (multigraph, respectively) G with maximum degree $\Delta(G)$, that guarantees a performance ratio of $1+\frac{1}{\Delta(G)}$ ($\frac{3}{2}$, respectively).*

Theorem 2. *[9] There is a simple greedy algorithm for coloring the vertices of any (multi)graph G with maximum degree $\Delta(G)$, with at most $\Delta(G)+1$ colors (i.e. $\chi(G) \leq \Delta(G)+1$).*

Corollary 2. *For any (multi)graph G, $\chi(G) \leq \chi'(G)+1$.*

Let $B=(U \cup V, E)$ be a bipartite graph. The sets U and V are defined as *upper* and *lower set*, respectively, in view of the usual graphical representation of bipartite graphs, although – of course – they can be freely interchanged. Let Δ_U and Δ_V be the maximum degrees of the vertices in the upper and lower

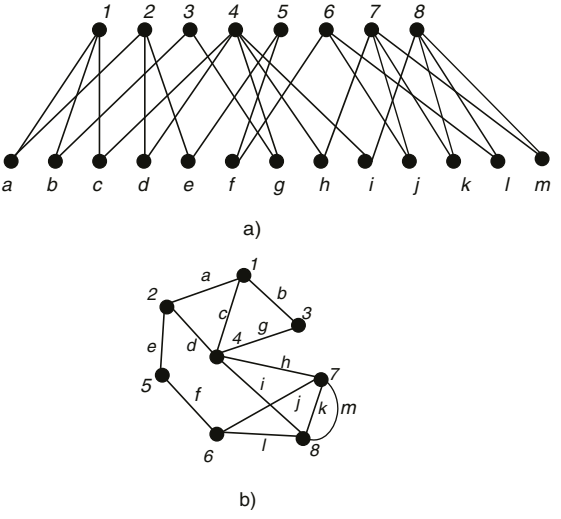


Fig. 1. a) A bipartite graph B; (b) Its incident graph $I(B)$

set, respectively, and let $\delta(x)$ denote the degree of a vertex x. We introduce the following three structures.

The first one is a multigraph associated with a bipartite graph with one partition containing only vertices of degree exactly two.

Definition 2. *Let $B = (U \cup V, E)$ be a bipartite graph with one partition containing only vertices of degree exactly two (w.l.o.g. let it be V). The* incidence graph $I(B) = (U, E')$ *is a multigraph defined as follows:*

i. *The vertex set corresponds to the upper set U of B;*
ii. *The edge set E' corresponds to the lower set V of B. For every vertex $e \in V$, such that $(u, e) \in E \wedge (v, e) \in E$, there exists an edge $(u, v) \in E'$.*

The incidence graph of the bipartite graph in Fig. 1 (a) is shown in Fig. 1 (b).

It is straightforward to see that all vertices in $I(B)$ have the same degree as they have in the upper set of B. Observe that any (multi)graph is the incidence graph of a bipartite graph with all vertices in the lower set of degree 2.

The second structure is a generalization of the incidence graph, extended to even-degree bipartite graphs. Assume that each vertex x of the lower set V in the bipartite graph $B = (U \cup V, E)$ has even degree $\delta(x)$, and that an ordering of the edges incident at each vertex is given. Let $\langle e_1^x, e_2^x, \ldots, e_{\delta(x)}^x \rangle$ denote the ordered sequence of edges incident at $x \in V$.

Definition 3. *Given a bipartite graph $B = (U \cup V, E)$ as above, the* extended incidence graph $Ext(B) = (U, E'')$ *of B is a (multi)graph defined as follows:*

– *the vertex set corresponds to the upper set U of B;*
– *for every vertex $x \in V$ and each couple $e_{2j+1}^x = \{x, u\} \in E$ and $e_{2j+2}^x = \{x, v\} \in E$, where $j \in \{0, \ldots, \delta(x)/2 - 1\}$, there exists an edge $(u, v) \in E''$.*

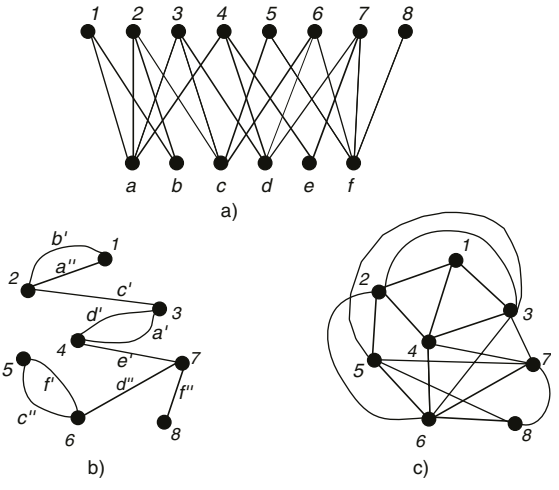


Fig. 2. (a) A bipartite graph B; (b) The extended incidence graph $Ext(B)$; (c) The node graph $N(B)$

Fig. 2 (b) shows the extended incidence graph of the graph in Fig. 2 (a).

Each vertex of $Ext(B)$ maintains the same degree it has in B, therefore $\Delta(Ext(B)) = \Delta(B)$. Furthermore, observe that a vertex $x \in V$ generates in $Ext(B)$ a set of $\delta(x)/2$ edges, denoted *Set(x)*. In Fig. 2 (b), $Set(a) = \{a', a''\}$, $Set(b) = \{b'\}$, $Set(c) = \{c', c''\}$, $Set(d) = \{d', d''\}$, $Set(e) = \{e'\}$, and, $Set(f) = \{f', f''\}$.

Note that, the extended incident graph represents a hypergraph where each $v \in V$ is a hyperedge.

The third structure is associated to a general bipartite graph and represents nodes in the upper set and the relations they have through nodes in the lower set.

Definition 4. *The* node-graph $N(B) = (U, E''')$ *of a bipartite graph* $B = (U \cup V, E)$ *is a simple graph defined as follows:*

- *the vertex set corresponds to the upper set* U *of* B*;*
- *an edge* $(u, v) \in E'''$ *if and only if there exists a vertex* x *in the lower set of* B *such that* $(u, x) \in E$ *and* $(x, v) \in E$*.*

In Fig. 2 (c) the node-graph of the graph in Fig. 2 (a) is depicted. The maximum degree $\Delta(N(B))$ of $N(B)$ is bounded by $min\{\Delta_U \cdot (\Delta_V - 1), |U|\}$.

3 Bipartite Graphs: Lower Bound

In [5], Bodlaender et al. proved that for every $\Delta \geq 2$, there is a bipartite graph with maximum degree Δ such that $\lambda^*_{0,1} \geq \frac{\Delta^2}{4}$. Since $\lambda^*_{0,1} \leq \lambda^*_{h,1}$, for any $h > 0$, the value $\frac{\Delta^2}{4}$ represents a lower bound for $\lambda^*_{h,k}$ of bipartite graphs. We improve this bound to $\Delta^2 - \Delta + 1$.

Definition 5. *Let $\mathcal{B}$ be the class of bipartite graphs $B = (U \cup V, E)$ satisfying the following conditions:*

a. Both the upper set U and the lower set V have $q^2 + q + 1$ vertices, for any integer $q \geq 0$;
b. Any vertex $u \in U \cup V$ has exactly degree $\Delta = q + 1$;
c. Given $u, v \in U$ (or, $u, v \in V$), then $|Adj(v) \cap Adj(u)| = 1$.

We show that the cardinality of $\mathcal{B}$ is infinite. Let us consider a projective plane $\mathcal{P}$ of order q, for some prime power q. Let U represent points and V lines of $\mathcal{P}$. For $u \in U$ and $v \in V$, $(u, v) \in E$ if and only if point u belongs to line v. It is straightforward to verify that the bipartite graph so defined satisfies the constraints in Definition 5 and, since all vertices in U (or in V) are at distance 2, we have $\lambda^*_{0,1}(B) \geq |U| = q^2 + q + 1 = \Delta^2 - \Delta + 1$.

Note that the class $\mathcal{B}$ has been introduced in [12] without deriving the lower bound. It is easy to see that this lower bound applies to s-partite graphs, for any s.

4 First Approximation Algorithm for Bipartite Graphs

The first algorithm we propose reduces the $L(h,k)$-labelling problem to both the edge- and the vertex-coloring of the (extended) incidence and the node-graphs associated with a given bipartite graph.

To make the exposition easier, the presentation proceeds in three steps. First we analyze bipartite graphs with vertices in the lower partition of regular degree 2. Then, we extend the technique to bipartite graphs with even-degree lower set. Finally, we generalize the algorithm to any bipartite graph.

4.1 Lower Set of Regular Degree 2

Let $B = (U \cup V, E)$ be a bipartite graph with all vertices in the lower set having regular degree 2. Let $I(B) = (U, E')$ be the corresponding incidence graph.

Let c_1 and c'_1 be two functions for vertex- and edge-coloring $I(B)$, requiring $\chi_1(I(B))$ and $\chi'_1(I(B))$ colors, respectively.

These two colorings can be exploited to deduce an $L(h,k)$-labelling of B in the following way:

- label each vertex v in the upper set of B with $k(c_1(v) - 1)$;
- label each vertex e in the lower set B with $(\chi_1(I(B)) - 1)k + h + k(c'_1(e) - 1)$,

where the term -1 is due to the fact that the smallest value for c_1 and c'_1 is 1, while for the $L(h,k)$-labelling it is 0. The term $(\chi_1(I(B)) - 1)k$ represents the largest color used in the upper set, and term h is the separation value.

It is easy to prove that the labelling produced is indeed an $L(h,k)$-labelling. The span of this labelling is $k(\chi_1(I(B)) + \chi'_1(I(B))) + h - 2k$.

Hence, from Corollary 2, we have:

$$\lambda(B) \leq 2k\chi'_1(I(B)) + h - k. \tag{1}$$

Consider now an optimal $L(h,k)$-labelling of B and call $f(v)$ the color assigned to vertex v in B. We can use this labelling to achieve a feasible edge-coloring for $I(B)$: consider each vertex e in the lower set of B and its corresponding edge e of $I(B)$; label e in $I(B)$ with $\lfloor f(e)/k \rfloor + 1$.

This labelling is a feasible edge-coloring of $I(B)$ since, for any pair of adjacent edges e and e' in $I(B)$, they are at distance two in V (via the vertex in the upper set corresponding to the common end-point) and, hence, they have labels at least k apart, so $\lfloor f(e)/k \rfloor + 1$ and $\lfloor f(e')/k \rfloor + 1$ are different. Therefore, for the number $\chi'_2(I(B))$ of used colors it holds:

$$\chi'^{*} \leq \chi'_2(I(B)) \leq \frac{\lambda^*(B)}{k} + 1 \tag{2}$$

Furthermore, if we consider any non trivial bipartite graph B, i.e. a connected graph with at least three vertices, with maximum degree $\Delta(B)$ then $\lambda^*(B) \geq h + (\Delta(B) - 1)k$. Since, in this case, $\Delta(B) \geq 2$, we have $\lambda^*(I(B)) \geq h + k$. On the other hand, since $k \leq h$, then $k \leq \lambda^*(I(B))/2$.

From Equation 1, Corollary 1 and Equation 2, and from the above observations, we get:

$$\lambda(B) \leq 3\lambda^*(B) + 2k + h \leq \frac{9}{2}\lambda^*(B).$$

The previous reasonings lead us to the following theorem:

Theorem 3. *The $L(h,k)$-labelling problem on bipartite graphs with a partition of regular degree 2 is $\frac{9}{2}$–approximable.*

4.2 Lower Set of Even Degree

In this subsection we extend the results of the previous section to bipartite graphs having vertices in the lower set of even degree.

Let $B = (U \cup V, E)$ be a bipartite graph, with vertices in the lower set of even degree. Let Δ_U and Δ_V be the maximum degree of the upper and lower set, respectively. Consider the associated node graph $N(B)$ and the extended incidence graph $Ext(B)$.

Observe that there exists a vertex-coloring function of $N(B)$ using at most $\chi(N(B)) \leq \Delta(N(B)) + 1 \leq \Delta_U \cdot (\Delta_V - 1) + 1$ colors.

Consider now the trivial greedy algorithm to color edges of $Ext(B)$ that sequentially considers each *Set(v)* and assigns to all its edges the same smallest feasible color.

Lemma 1. *The greedy algorithm for edge-coloring $Ext(B)$ uses at most $\chi'(Ext(B)) \leq \Delta_V \cdot \Delta_U - \Delta_V + 1$ colors.*

Proof. Let us consider *Set(x)* for some $x \in V$. *Set(x)* has at most $\Delta_V/2$ vertex disjoint edges, each one of them incident at most $2(\Delta_U - 1)$ further edges; hence, to color *Set(x)* at most $2 \cdot \Delta_V/2 \cdot (\Delta_U - 1)$ colors must be avoided. The proof follows.

Corollary 3. *$Ext(B)$ can be edge-colored so that all edges in $Set(x)$, for any x, receive the same color, with a guaranteed performance ratio of Δ_V.*

Proof. The claim follows from Lemma 1 and from the obvious inequality $\chi'^*(Ext(B)) \geq \Delta(Ext(B)) = \Delta_U$.

An immediate consequence of the above result is shown in the following theorem.

Theorem 4. *Given an n–vertex hypergraph $\mathcal{H}$ of dimension d, then there exists an approximation algorithm coloring edges of $\mathcal{H}$ with guaranteed approximation ratio of d.*

We are now ready to derive a feasible $L(h,k)$-labelling of B as follows. Let c_1 be a vertex-coloring function for $N(B)$ and let c'_1 be an edge-coloring function for $Ext(B)$, with the property that edges of *Set(x)* have all the same color, for each x. Let $\chi_1(N(B))$ and $\chi'_1(Ext(B))$ denote the number of colors required.

Proceed as follows:

- label each vertex v in the upper set of B with $k(c_1(v) - 1)$;
- label each vertex x in the lower set of B with $(\chi_1(N(B)) - 1)k + h + k(c'_1(Set(x)) - 1)$.

By the definitions of $N(B)$, $Ext(B)$, and the specific edge-coloring function c'_1, the labelling obtained is feasible and its span is $\lambda(B) = k\chi_1(N(B)) + k\chi'_1(Ext(B)) + h - 2k$.

Reminding that $\chi_1(N(B)) \leq \Delta(N(B)) + 1 \leq \Delta_U(\Delta_V - 1) + 1$ and that $\chi'_1(Ext(B)) \geq \Delta_U$ we have:

$$\lambda(B) \leq k\chi'_1(Ext(B))\Delta_V + h - k. \tag{3}$$

Similarly to Subsection 4.1, we assume to have an optimal $L(h,k)$-labelling for B with span $\lambda^*(B)$ and deduce a feasible edge-coloring for $Ext(B)$ with the property that all edges in *Set(x)* have the same color, for any x. Let $\chi'_2(Ext(B))$ be the number of used colors. It follows that:

$$\chi'^*(Ext(B)) \leq \chi'_2(Ext(B)) \leq \frac{\lambda^*(B)}{k} + 1. \tag{4}$$

Considering that, for the class of graphs under consideration $\Delta \geq 3$ (if $\Delta = 2$ we have already given a result), we have $\lambda^*(B) \geq h + 2k$. Additionally, since $h \geq k$, then $k \leq \frac{\lambda^*(B)}{3}$. From Equation 3, Corollary 3, Equation 4, and the above observations, we have:

$$\begin{aligned} \lambda(B) &\leq k\left(\frac{\lambda^*(B)}{k} + 1\right)\Delta_V^2 + h - k \leq \\ &\leq \Delta_V^2\lambda^*(B) + (\Delta_V^2 - 3)\frac{\lambda^*(B)}{3} + \lambda^*(B) = \\ &= \frac{4}{3}\Delta_V^2\lambda^*(B). \end{aligned} \tag{5}$$

From the above discussion we have the following theorem:

Theorem 5. *The $L(h,k)$-labelling problem on bipartite graphs with all vertices in the lower set of even degree is $\frac{4}{3}\Delta_V^2$–approximable, where Δ_V is the maximum degree of the lower set.*

4.3 General Bipartite Graphs

In this subsection we further extend the previous results obtaining an approximation algorithm guaranteeing a performance ratio of $\frac{4}{3}D^2$ for each bipartite graph where D is the smallest even value bounding the minimum of the maximum degrees of the two partition.

Let $B = (U \cup V, E)$ be a bipartite graph. W.l.o.g. let $\Delta_V = \min \Delta_U, \Delta_V$, hence D is either Δ_V or $\Delta_V + 1$. Consider all vertices in the lower set with odd degree, and for each such vertex x, add a dummy vertex v_x in the upper set and a dummy edge (x, v_x) in E. In this way a new bipartite graph $B' = (U' \cup V, E')$ is generated and its lower set has maximum degree D and all vertices in V have even degree. Hence, we can consider graphs $N(B)$ and $Ext(B')$ and apply Theorem 5. Indeed, it is easy to see that a feasible $L(h,k)$-labelling for B' is a $L(h,k)$-labelling for B, also. Obviously, the viceversa could not be true. Hence, we have the following theorem:

Theorem 6. *The $L(h,k)$-labelling problem on bipartite graphs is $\frac{4}{3}D^2$–approximable, where D is the smallest even value bounding the minimum of the maximum degrees of the two partitions.*

5 Second Approximation Algorithm

In this section we propose another approximation algorithm for $\lambda^*_{h,k}(B)$ on a general bipartite graph B with a ratio $\min(h, 2k)\sqrt{n} + o(k\sqrt{n})$, not depending on the degree of the graph.

In [14] the author proves the existence of an approximation algorithm for vertex-coloring the square G^2 of any n vertex graph G with performance ratio $\sqrt{n-1} + 1$.

Consider any bipartite graph $B = (U \cup V, E)$. We remind that any $L(1,1)$-labelling of B is a vertex-coloring of B^2, and it partitions the vertex set in classes such that vertices in different classes have different colors. Furthermore, if a class contains nodes of both U and V, we can split it into two classes so that each of them contains only elements in the upper (lower) set. Let $L_{1,1}(U)$ and $L_{1,1}(V)$ be the number of classes covering U and V, respectively. It holds $L_{1,1}(U) + L_{1,1}(V) \leq 2(\lambda_{1,1}(B) + 1)$, where the term $+1$ derives from the fact that the smallest color of an $L(1,1)$-labelling is 0. Additionally, observe that $\lambda^*_{h,k}(B) \geq \lambda^*_{1,1}(B)$. We are now ready to describe the algorithm.

Run algorithm described in [14] on B to obtain an $L(1,1)$-labelling such that:

$$\frac{\lambda_{1,1}(B)}{\lambda^*_{1,1}(B)} \leq \sqrt{n-1} + 1.$$

If $2k \leq h$, consider the classes induced by colors and split them into classes separating vertices of U and of V.

Number all classes in U, starting from 0, and all classes in V, starting again from 0. Then, for each vertex $v \in U$ belonging to class numbered $f(v)$, label v with $kf(v)$. Finally, for each vertex $v \in V$ belonging to class numbered $f(v)$, label v with $k(L_{1,1}(U)-1)+h+kf(v)$. The computed labelling is a feasible $L(h,k)$-labelling of B and has span $\lambda_{h,k}(B) \leq k(L_{1,1}(U)-1)+k(L_{1,1}(V)-1)+h \leq 2k\lambda_{1,1}(B)+h$. The performance ratio of the previous algorithm is:

$$\frac{\lambda_{h,k}(B)}{\lambda^*_{h,k}(B)} \leq \frac{2k\lambda_{1,1}(B)+h}{\lambda^*_{h,k}(B)} \leq$$

$$\leq \frac{2k\lambda_{1,1}(B)}{\lambda^*_{1,1}(B)} + \frac{h}{\lambda^*_{h,k}(B)} \leq 2k(\sqrt{n-1}+1)+1$$

as $\lambda^*_{h,k}(B)$ is at least h for each non trivial graph with at least two vertices.

If $2k > h$, instead of labelling nodes as above described, we proceed as follows: Consider the classes induced by colors and label each node in the class colored $f(v)$ with $hf(v)$. The produced labelling is feasible and its span is $\lambda_{h,k}(B) \leq h\lambda_{1,1}(B)$. Hence, the performance ratio is $h\sqrt{n-1}+h$.

The above discussion leads to the following theorem:

Theorem 7. *Given a n-vertex bipartite graph B, there exists a polynomial time approximation algorithm for computing an $L(h,k)$-labelling of B with $min(h,2k)\sqrt{n}+o(k\sqrt{n})$ guaranteed performance ratio.*

Observe that the $\frac{4}{3}D^2$-approximation algorithm is better than this one when $D = O(n^{1/4})$.

Furthermore, it is easy to generalize the above strategy to s-partite graph. In this case we have the following theorem:

Theorem 8. *Given an n-vertex s-partite graph G, there exists a polynomial time approximation algorithm for computing an $L(h,k)$-labelling of G with $min(h,sk)\sqrt{n}+o(sk\sqrt{n})$ guaranteed performance ratio.*

Proof. The proof easily derives by generalizing Theorem 7, and it is omitted in this extended abstract for the sake of brevity.

6 General Graphs

In this section, we show a result stating the strong tie between the $L(h,k)$-labelling problem and the problem of coloring the vertices of the square of a graph with the minimum number of colors.

Theorem 9. *Let be given any value $\alpha > 1$ and a graph G. If there exists an algorithm finding an approximate vertex-coloring of G^2 with approximation*

ratio $\frac{\chi(G^2)}{\chi^*(G^2)} \leq \alpha$*, then there exists an algorithm finding an approximate* $L(h,k)$*-labelling of* G *with approximation ratio* $\frac{\lambda_{h,k}(G)}{\lambda^*_{h,k}(G)} \leq h\alpha$*.*

Conversely, let be given any value $\beta > 1$ *and a graph* G*. If there exists an algorithm finding an approximate* $L(h,k)$*-labelling with approximation ratio* $\frac{\lambda_{h,k}(G)}{\lambda^*_{h,k}(G)} \leq \beta$*, then there exists an algorithm finding an approximate vertex-coloring of* G^2 *with approximation ratio* $\frac{\chi(G^2)}{\chi^*(G^2)} \leq h\beta$*.*

Proof. Suppose there exists an algorithm coloring vertices of G^2 with performance ratio $\frac{\chi(G^2)}{\chi^*(G^2)} \leq \alpha$, for any graph $G = (V, E)$. Let $f(v)$ be the color assigned to vertex v. Then a feasible $L(h,k)$-labelling for G is obtained by assigning label $h(f(v) - 1)$ to v, assuming $h \geq k$. It is easy to see that such a labelling is a feasible $L(h,k)$-labelling and its performance ratio is:

$$\frac{\lambda_{h,k}(G)}{\lambda^*_{h,k}(G)} \leq \frac{h(\lambda_{1,1}(G))}{\lambda^*_{1,1}(G)} \leq h\alpha.$$

Conversely, suppose there exists an approximation algorithm for $L(h,k)$-labelling with performance ratio $\frac{\lambda_{h,k}(G)}{\lambda^*_{h,k}(G)} \leq \beta$ for any graph G.

Since $h \geq k \geq 1$, a $L(h,k)$-labelling is always a feasible $L(1,1)$-labelling for G. On the other hand, using a similar reasoning as in Section 3, $\lambda^*_{1,1}(G) \geq \lambda^*_{h,k}/h$. It follows that:

$$\frac{\chi(G^2)}{\chi^*(G^2)} \leq \frac{h\lambda_{h,k}(G)}{\lambda^*_{h,k}(G)} \leq h\beta$$

Corollary 4. *The problem of vertex-coloring the square of a graph with the minimum number of colors is in APX if and only if the problem of* $L(h,k)$*-labelling a graph is in APX, for each constant value* h *and* $k \leq h$*.*

Note that for a not constant h, the $L(h,k)$-labelling problem is not easier than the vertex-coloring of the square of a graph.

From Theorem 9 and considering the approximation algorithm described in [14], we can state the following theorem:

Theorem 10. *Given a* n*-vertex graph* G*, there exists a polynomial time approximation algorithm for computing an* $L(h,k)$*-labelling of* G *with* $h(\sqrt{n-1}+1)$ *guaranteed performance ratio.*

References

1. K. I. Aardal, S. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and solutions techniques for frequency assignmet problems. Technical report, Konrad-Zuse-Zentrum *für* Informationtechnik Berlin, December 2001.
2. G. Agnarsson, R. Greenlaw, and M.M. Halldorson. On powers of chordal graphs and their colorings. *Congr. Numerant.*, 144:41–65, 2000.

3. G. Agnarsson and M. M. HalldLorsson. Coloring powers of planar graphs. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 654.662, N.Y., January 9–11 2000. ACM Press.
4. N. Alon and J. H. Kim. On the degree, size, and chromatic index of a uniform hypergraph. *J. Combin. Theory Ser. A*, 77:165–170, 1997.
5. H. L. Bodlaender, T. Kloks, R. B. Tan, and J. van Leeuwen. λ-coloring of graphs. *Proc. of STACS 2000. Lecture Notes in Computer Science*, 1770:395–406, 2000.
6. T. Calamoneri. Exact solution of a class of frequency assignment problems in cellular networks (extended abstract). *Proc. of the ICTCS 2003. Lecture Notes in Computer Science*, 2841:163–173, 2003.
7. T. Calamoneri. The l(h, k)-labelling problem: A survey. Technical Report 04/2004, Dept. of Computer Science, University of Rome *La Sapienza*, 2004.
8. T. Calamoneri, A. Pelc, and R. Petreschi. Labelling trees with a condition at distance two. In *Proc. of R.C. Bose Centenary Symp. on Discr. Math. and Applications*, 2002. Accepted to Discrete Mathematics.
9. R. Diestel. Graph theory. Springer-Verlag, New York, 1997. Translated from the 1996 German original.
10. J. P. Georges and D. W. Mauro. Some results on λ_k^j-numbers of the products of complete graphs. *Congr. Numer.*, 140:141–160, 1999.
11. J. P. Georges, D. W. Mauro, and M. I. Stein. Labeling products of complete graphs with a condition at distance two. *SIAM Journal on Discrete Mathematics*, 14(1):28–35, February 2001.
12. J. R. Griggs and R. K. Yeh. Labelling graphs with a condition at distance 2. *SIAM Journal on Discrete Mathematics*, 5(4):586–595, November 1992.
13. Y. L. Lin and S. S. Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8(1):99–118, February 1995.
14. S. T. McCormick. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Mathematical Programming*, 26(2):153–171, 1983.
15. R. A. Murphey, P. M. Pardalos, and M. G. C. Resende. Frequency assignment problems. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of combinatorial optimization*, volume Supplement A. Kluwer Academic Publishers, 1999.
16. N. Pippinger and J. Spencer. Asymptotic behaviour of the chromatic index for hypergraphs. *J. Combin. Theory Ser. A*, 51:24–42, 1989.
17. C. E. Shannon. A theorem on coloring the lines of a network. *J. Math. Phys*, 28:148–151, 1949.
18. J. van den Heuvel, R. A. Leese, and M. A. Shepherd. Graph labeling and radio channel assignment. *Journal of Graph Theory*, 29, 1998.
19. V. G. Vizing. On an estimate of the chromatic class of a p-graph (in russian). *Diskret. Analiz*, 3:23–30, 1964.

A Tight Bound for Online Coloring of Disk Graphs*

Ioannis Caragiannis[1], Aleksei V. Fishkin[2], Christos Kaklamanis[1], and Evi Papaioannou[1]

[1] Research Academic Computer Technology Institute and Department of Computer Engineering and Informatics, University of Patras, 26500 Rio, Greece
[2] Max Planck Institut für Informatik, Stuhlsatzenhausweg 85, Geb. 46.1, 66123 Saarbrücken, Germany

Abstract. We present an improved upper bound on the competitiveness of the online coloring algorithm First-Fit in disk graphs which are graphs representing overlaps of disks on the plane. We also show that this bound is best possible for deterministic online coloring algorithms that do not use the disk representation of the input graph. We also present a related new lower bound for unit disk graphs.

1 Introduction

We study minimum coloring, a fundamental combinatorial optimization problem in graphs. Given a graph G, the minimum coloring problem is to find an assignment of colors (denoted by positive integers) to the nodes of the graph so that no two nodes connected by an edge are assigned the same color and the number of colors used is minimized. We consider intersection graphs modeling overlaps of disks on the plane.

The intersection graph of a set of disks in the Euclidean plane is the graph having a node for each disk and an edge between two nodes if and only if the corresponding disks overlap. Each disk is defined by its radius and the coordinates of its center. Two disks overlap if the distance between their centers is at most equal to the sum of their radii. A graph G is called a *disk graph* if there exists a set of disks in the Euclidean plane whose intersection graph is G. The set of disks is called the disk representation of G. A disk graph is called *unit disk graph* if all disks in its disk representation have the same radius. A disk graph is σ*-bounded* if the ratio between the maximum and the minimum radius among all the disks in its disk representation is at most σ.

In disk graphs, minimum coloring is important since it can model frequency assignment problems in radio communication networks utilizing the Frequency Division Multiplexing technology [10]. Consider a set of transmitters located in

* This work was partially supported by the European Union under IST FET Project CRESCCO.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 78–88, 2005.

fixed positions within a geographical region. Each transmitter may select to use a specific frequency from an available spectrum in order to transmit its messages. Two transmitters can successfully (i.e., without signal interference) transmit messages simultaneously either if they use different frequencies or if they use the same frequency and their ranges do not overlap. Given a set of transmitters in a radio network, in order to guarantee successful transmissions simultaneously, the important engineering problem to be solved is the frequency assignment problem where the objective is to minimize the number of frequencies used all over the network. Assuming that all transmitters have circular range, the graph reflecting possible interference between pairs of transmitters is a disk graph. The frequency assignment problem is equivalent to minimum coloring.

An instance of the minimum coloring problem may or may not include the disk representation (i.e., disk center coordinates and/or radii) of the disk graph as part of the input. Clearly, the latter case is more difficult. Information about the disk representation of a disk graph is not easy to extract. Actually, determining whether a graph is a disk graph is an NP-complete problem [11].

The minimum coloring problem has been proved to be NP-hard in [3, 8] even for unit disk graphs. A naive algorithm is algorithm First-Fit: it examines the nodes of the graph in an arbitrary order and assigns to each node the smallest color not assigned to its already examined neighbors. Clearly, First-Fit does not use the disk representation. It computes 5-approximate solutions in unit disk graphs [7, 14]. By processing the nodes of the graph in a specific order, First-Fit computes 3-approximate solutions in unit disk graphs [8, 14, 15]. In general disk graphs, a smallest-degree-last version of First-Fit achieves an approximation ratio of 5 [7, 13, 14].

In the online versions of the problem, the disk graph is not given in advance but is revealed in steps. In each step, a node of the graph appears together with its edges incident to nodes appeared in previous steps (and possibly, together with the center coordinates and/or the radius of the corresponding disk). When a node appears, an online coloring algorithm decides which color to assign to the node. The decisions of the algorithm at a step cannot change in the future.

The performance of an online algorithm is measured in terms of its competitive ratio (or competitiveness, [1]) which is defined as the maximum over all possible sequences of disks of the ratio of the number of colors used by the algorithm over the minimum number of colors sufficient for coloring the graph (i.e., its chromatic number).

First-Fit is essentially an online algorithm. It has been widely studied in a more general context and has been proved to be $\Theta(\log n)$-competitive in inductive graphs with n nodes [12, 9]. Disk graphs are inductive [4, 5] so the upper bound holds for disk graphs as well. The lower bound holds also for trees (which are disk graphs) so the $\Theta(\log n)$ bound holds for general disk graphs. In unit disk graphs, First-Fit is at most 5-competitive [7, 14] while for σ-bounded disk graphs with n nodes, it is at most $O(\min\{\log n, \sigma^2\})$-competitive [4]. For unit disk graphs, a lower bound of 2 on the competitiveness of any deterministic online coloring algorithm is presented in [6]. The best known lower bound

on the competitiveness of deterministic coloring algorithms in σ-bounded disk graphs is $\Omega(\min\{\log n, \log\log\sigma\})$ [4]. A competitive ratio of $O(\min\{\log n, \log\sigma\})$ is achieved by two algorithms presented in [4] and [2]. The former uses the disk representation while the latter does not but it is quite impractival. Both algorithms use First-Fit as a subroutine.

In this paper we show that algorithm First-Fit itself is $O(\log\sigma)$-competitive when applied to σ-bounded disk graphs. This significantly improves the previously known upper bound of $O(\sigma^2)$ on the competitiveness of First-Fit. Furthermore, it matches the best known upper bound for online deterministic coloring algorithms, previously achieved either by algorithms that use the disk representation [4] or by quite impractical algorithms that do not use the disk representation [2]. Our second result indicates that First-Fit has optimal competitiveness (within constant factors) among all deterministic online algorithms for disk graphs that do not use the disk representation. In particular, we show that any deterministic online coloring algorithm that does not use the disk representation has competitive ratio $\Omega(\log\sigma)$ on σ-bounded disk graphs. Combined with previous results, our lower bound establishes a tight bound of $\Theta(\min\{\log n, \log\sigma\})$ on the optimal competitiveness of deterministic online coloring algorithms in σ-bounded disk graphs with n nodes that do not use the disk representation. We also prove a new lower bound of 2.5 on the competitiveness of deterministic online coloring algorithms for unit disk graphs that do not use the disk representation. This result improves a previous lower bound of 2 [6].

The rest of the paper is structured as follows. A discussion on previous upper bounds and the proof of the upper bound on the competitiveness of First-Fit are presented in Section 2. The lower bounds are presented in Section 3. We conclude with open problems in Section 4.

2 The Upper Bound

In this section, we prove the upper bound for algorithm First-Fit. Although this upper bound can also be achieved by two other known algorithms presented in [4] and [2], respectively, our result is important because of the simplicity of algorithm First-Fit.

The algorithm of Erlebach and Fiala [4] classifies the disks into a logarithmic number of classes so that the disks belonging to the same class form a 2-bounded disk graph and runs algorithm First-Fit in each class using disjoint sets of colors for coloring the disks of different classes. The classification is performed according to the radii of the disks; hence, the algorithm uses the disk representation. The proof of the $O(\log\sigma)$ upper bound follows by the fact that algorithm First-Fit has constant competitive ratio on 2-bounded disk graphs.

The algorithm Layered classifies the disks into layers and applies algorithm First-Fit to each layer separately, using a different set of colors in each layer. Layers are numbered with integers $1, 2, ...$ and a disk is classified into the smallest layer possible under the constraint that it cannot be classified into a layer if it overlaps with at least 16 mutually non-overlapping disks belonging to this layer.

The proof that algorithm Layered is $O(\log \sigma)$-competitive is based on the following arguments. First, if a disk of radius R belongs to some layer $i > 1$, then there is a disk of radius at most $R/2$ belonging to layer $i - 1$. Hence, if the disk graph given as input to algorithm Layered is σ-bounded, the number of layers is at most $1 + \log \sigma$. The logarithmic competitive ratio follows since the maximum independent set in the neighborhood of each node within each layer has size at most 15, and, hence, algorithm First-Fit is proved to have constant competitive ratio within each layer. Clearly, algorithm Layered does not use the disk representation.

For checking whether a new node presented has 16 or more non-overlapping disks of some layer in its neighborhood may require time $\Omega(n^{16})$. This could be decreased to $\Omega(n^8)$ by changing the constraint so that a disk cannot be classified into a layer if it overlaps with at least 8 mutually non-overlapping disks belonging to this layer. Still, it can be proved that there exists a constant $\alpha > 1$ such that for each disk of radius R belonging to some layer $i > 1$, there exists a disk at layer $i - 1$ of radius smaller than R/α. This is the best possible improvement in the idea of algorithm Layered since, for any $\alpha > 1$ arbitrarily close to 1 (e.g., $\alpha = 1 + 1/\sigma$), a disk of radius R can overlap with 7 mutually non-overlapping disks of radius R/α, and, hence, the logarithmic upper bound on the number of layers cannot be established.

Surprisingly, we show that algorithm First-Fit itself is at most $O(\log \sigma)$-competitive, improving the previously known $O(\sigma^2)$ upper bound. Combining this result with the $O(\log n)$ upper bound which is known for the competitive ratio of First-Fit we obtain that First-Fit is $O(\min\{\log n, \log \sigma\})$-competitive. Algorithm First-Fit runs in time proportional to the number of edges of the disk graph, i.e., $O(n^2)$, and does not use the disk representation. Hence, it is much simpler than the previously known algorithms that achieve the same bounds.

Theorem 1. First-Fit *is* $O(\log \sigma)$*-competitive for* σ*-bounded disk graphs.*

Proof. Let G be a σ-bounded disk graph with chromatic number κ. Assume that the nodes of G appear online and are colored by algorithm First-Fit. Consider a representation of G by overlapping disks on the plane of radii between r (the radius of the smallest disk) and $R_{\max}$ (the radius of the largest disk) so that $R_{\max}/r \leq \sigma$. We classify the nodes into levels $0, 1, ..., \lfloor \log (R_{\max}/r) \rfloor$ as follows: a node corresponding to a disk of radius R belongs to level $\lfloor \log (R/r) \rfloor$. Since $R_{\max}/r \leq \sigma$, the index of the last level is at most $\lfloor \log \sigma \rfloor$.

We will first show that a node of G belonging to level $i \geq 0$ is adjacent to at most $15(\kappa - 1)$ other nodes of level at least i.

Assume otherwise that there exists a node u of G at level i which is adjacent to at least $15\kappa - 14$ other nodes of level at least i. Let R be the radius of the disk d corresponding to node u in the disk representation. Then $i = \lfloor \log (R/r) \rfloor$. Also, let S_d be the set of disks corresponding to nodes adjacent to d which belong to levels at least i. Clearly, all the disks of S_d have radii at least $r2^i$.

We apply the following shrinking procedure on the disks of S_d. We shrink each disk d' in S_d into a disk of radius $r2^{\lfloor \log R/r \rfloor}$ as follows: If the center $c_{d'}$ of d' is inside d, we shrink d' into a disk having the same center $c_{d'}$. Otherwise, let

$p_{d'}$ be the point in the periphery of d' which is closest to the center of d. We shrink d' so that $p_{d'}$ is again the point in the periphery of d' which is closest to the center of d. Denote by S'_d the set of shrunk disks. Clearly, each of the disks in S'_d overlaps with d since either its center is contained in d or a point in its periphery is contained in d. This means that all disks in S'_d are completely contained into the disk of radius $R + 2^{i+1}$ centered at the center c_d of disk d. An example of the shrinking procedure is depicted in Figure 1.

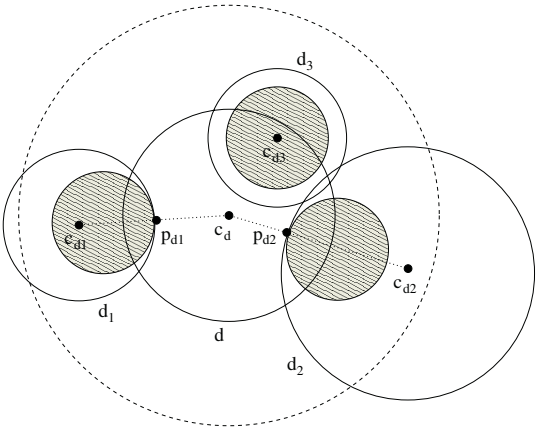


Fig. 1. The shrinking procedure. The disk d overlaps with three disks. Grey disks are the three corresponding shrunk disks

The node-induced subgraph H of G defined by the nodes of G corresponding to the disks of S_d is $(\kappa - 1)$-colorable since the graph G is κ-colorable and the nodes of H are all adjacent to u in G. Consequently, since by our assumption H contains the neighbors of u in G with levels at least i and since there are at least $15\kappa - 14$ such nodes, the maximum independent set in H has size at least 16. Consider such an independent set of 16 nodes in H and the 16 non-overlapping disks of S_d corresponding to these nodes. Clearly, the 16 corresponding shrunk disks of S'_d are also non-overlapping. Each of these disks has radius $r2^i$ and, hence, their total area is

$$\begin{aligned} 16\pi \left(r2^i\right)^2 &= \pi \left(r2^{i+1} + r2^{i+1}\right)^2 \\ &> \pi \left(R + r2^{i+1}\right)^2 . \end{aligned}$$

Since these disks are non-overlapping, this contradicts the fact that all disks of S'_d are completely contained in the disk of radius $R + r2^{i+1}$ centered at c_d. Consequently, our assumption is incorrect and u is adjacent to at most $15(\kappa - 1)$ other nodes of G of level at least i.

We will now show that each node of G at level $i \geq 0$ is colored by algorithm First-Fit with a color in the range $[1, (15\kappa - 14)(i + 1)]$. Hence, the maximum color that can be assigned to a node of G by First-Fit is at most

$(15\kappa - 14)\,(\lfloor \log \sigma \rfloor + 1)$. Since G has chromatic number κ, this implies that algorithm First-Fit is $O(\log \sigma)$-competitive.

We use induction on the level of nodes. The statement is true for nodes of level 0, since any such node is adjacent to at most $15(\kappa - 1)$ nodes of G and, hence, it will be assigned a color in $[1, 15\kappa - 14]$.

Now assume that the statement is true for nodes of level $i = 0, ..., k$ (for $k < \lfloor \log \sigma \rfloor$). We will show that it also holds for nodes of level $k+1$. Consider a node u at level $k+1$. This node will be adjacent to nodes of smaller levels which may use up to color $(15\kappa-14)(k+1)$ and to at most $15(\kappa-1)$ additional nodes of levels at least $k+1$. Hence, the maximum color that can be assigned by algorithm First-Fit to node u is $(15\kappa - 14)(k+1) + 15(\kappa - 1) + 1 = (15\kappa - 14)(k+2)$. This completes the proof of the theorem. □

3 The Lower Bound

The result proved in the following establishes that algorithm First-Fit achieves the best possible competitive ratio (within constant factors) among all deterministic online coloring algorithms that do not use the disk representation.

We present an adversary ADV which, on input an integer k and a deterministic online coloring algorithm A, outputs a tree T with at most 2^{k-1} nodes such that A colors T with at least k colors. We describe the adversary ADV in the following. This is a non-recursive description of the adversary very similar to that used in [4] for proving lower bounds on disk graphs and (in a more general form) in [12] for proving lower bounds on inductive graphs. We use the notation $\langle s, t \rangle$ to represent the nodes in the tree but the root, where $t \geq 1$ is an integer representing the level of the node and s a binary string of length $k - t - 1$. We use the function $str()$ which, on input integers $i \geq 0$ and j with $0 \leq j \leq 2^i$, returns a string of length i (possibly empty) which is the binary representation of j with i binary digits. We use the symbol $\odot$ to denote the concatenation of two strings. The adversary ADV can be described as follows.

Create 2^{k-2} nodes labeled as $\langle str(i, k-2), 1 \rangle$, for each $i = 0, ..., 2^{k-2} - 1$, and introduce them to algorithm A.
For $i = 2$ to $k - 1$
  For $j = 0$ to $2^{k-i-1} - 1$
    Let S_ℓ be the set of colors assigned by A to nodes

$$\langle str(j, k-i-1) \odot \overbrace{000...00}^{i-1 \text{ times}}, 1\rangle, \langle str(j, k-i-1) \odot \overbrace{00...00}^{i-2}, 2\rangle,$$
$$..., \langle str(j, k-i-1) \odot 0, i-1\rangle.$$

    Let S_r be the set of colors assigned by A to nodes

$$\langle str(j, k-i-1) \odot 1\overbrace{00...00}^{i-2}, 1\rangle, \langle str(j, k-i-1) \odot 1\overbrace{0..00}^{i-3}, 2\rangle,$$
$$..., \langle str(j, k-i-1) \odot 1, i-1\rangle.$$

If $S_\ell = S_r$ then
Create a new node $\langle str(j, k-i-1), i\rangle$ connected to nodes

$$\langle str(j,k-i-1) \odot 1\overbrace{00..00}^{i-2}, 1\rangle, \langle str(j,k-i-1) \odot 1\overbrace{0..00}^{i-3}, 2\rangle,$$

$$..., \langle str(j,k-i-1) \odot 1, i-1\rangle$$

and introduce it to algorithm A.
else
Let $\langle s, t\rangle$ be the node to which A assigns a color not in S_ℓ.
Rename it as $\langle str(j, k-i-1), i\rangle$
Endif
Endfor
Endfor
Create a new node r connected to nodes labeled

$$\overbrace{000...00}^{k-2}, 1\rangle, \langle\overbrace{000...00}^{k-3}, 2\rangle, ..., \langle 0, k-2\rangle, \langle\emptyset, k-1\rangle,$$

and introduce it to algorithm A.

The adversary forces the algorithm to use at least k colors. In each iteration, it can be shown by induction on i that all the $i-1$ nodes examined for defining the set S_ℓ (and similarly for S_r) are colored with $i-1$ different colors. Hence, after the if-then-else statement, the adversary will have forced algorithm A to use i different colors, i.e., $k-1$ colors at the end of all iterations. This clearly holds if the sets S_ℓ an S_r are not the same (else statement). Otherwise, it is guaranteed by the introduction of a new node which is connected to nodes colored with the $i-1$ different colors of S_r (if-then statement). Then, the last node r is connected to nodes with $k-1$ different colors and will be assigned a k-th color by algorithm A.

Also, it can be seen that when a new node is introduced in an iteration, the nodes to which it is connected will not be connected to other nodes in subsequent iterations. Hence, in general, the resulting graph is a forest. The number of nodes is at most 2^{k-1} since there are 2^{k-2} nodes at level 1, at most one new node in each iteration and one more node at the end. Actually, when the adversary runs against algorithm First-Fit, then the constructed graph is a tree with exactly 2^{k-1} nodes (in each iteration, a new node is introduced). We denote this tree by $T_{FF}(k)$ and we will first show that this is an α^{k-1}-bounded disk graph, for every $\alpha > 2$. Then, we will show how to adapt this construction for forests produced by the adversary against other deterministic online coloring algorithms.

Given a disk d of radius R corresponding to some node of the tree $T_{FF}(k)$, we define the vertical stripe of d to be the vertical stripe of width $2R$ which completely contains d. In our construction, the disk representation of $T_{FF}(k)$ is such that the disks corresponding to nodes in the subtree of a node u do

not cross the boundaries of the vertical stripe of the disk d corresponding to u. Furthermore, the vertical stripes of any two disks corresponding to children of the same node are disjoint. These two invariants guarantee that the disks corresponding to nodes belonging to different subtrees do not overlap.

We first locate a disk of radius α^{k-1} corresponding to the root r of the tree. Disks corresponding to nodes of level i (for $i = 1, ..., k-1$) will have radius α^{i-1}.

A node u at level i with $i = 2, ..., k$, has $i-1$ children u_1, ..., u_{i-1} in $T_{FF}(k)$ with levels $1, ..., i-1$, respectively. Let d be the disk corresponding to node u and let $d_1, ..., d_{i-1}$ be the disks corresponding to its children u_1, ..., u_{i-1}, respectively. Assuming that the center of the disk d has horizontal coordinate h, the center of the disk d_j has horizontal coordinate $h_j = h - \alpha^{i-1} + 3\alpha^{i-1}2^{-i+j}$. The horizontal stripes of the disks $d_1, ..., d_{i-1}$ are disjoint since for two disks d_j and $d_{j'}$ with $j > j'$, their centers differ in the horizontal coordinate by

$$\begin{aligned} h_j - h_{j'} &= (h - \alpha^{i-1} + 3\alpha^{i-1}2^{-i+j}) - (h - \alpha^{i-1} + 3\alpha^{i-1}2^{-i+j'}) \\ &= \frac{3}{2}\alpha^{j-1}(\alpha/2)^{i-j}(2 - 2^{-j+j'+1}) \\ &> \frac{3}{2}\alpha^{j-1} \\ &> \alpha^{j-1} + \alpha^{j'-1} \end{aligned}$$

which is the sum of their radii.

Furthermore, neither the leftmost disk d_1 nor the rightmost disk d_{i-1} cross the boundary of the horizontal stripe of d. Indeed, the leftmost point of d_1 has horizontal coordinate

$$\begin{aligned} h_1 - 1 &= h - \alpha^{i-1} + 3\alpha^{i-1}2^{-i+1} - 1 \\ &= h - \alpha^{i-1} + 3(\alpha/2)^{i-1} - 1 \\ &> h - \alpha^{i-1} + 2 \\ &> h - \alpha^{i-1} \end{aligned}$$

which is the horizontal coordinate of the left boundary of the horizontal stripe of d. Also, the rightmost point of d_{i-1} has horizontal coordinate

$$\begin{aligned} h_{i-1} + \alpha^{i-2} &= h - \alpha^{i-1} + 3\alpha^{i-1}/2 + \alpha^{i-2} \\ &= h + \alpha^{i-1}/2 + \alpha^{i-2} \\ &< h + \alpha^{i-1} \end{aligned}$$

which is the horizontal coordinate of the right boundary of the horizontal stripe of d.

The vertical coordinate of the center of disk d_j is defined so that it is smaller than the vertical coordinate of the lowest point in the intersection of disk d_j with disk d. This guarantees that, among all disks corresponding to nodes in the subtree of u, the disks that d overlaps with are those corresponding to its children in $T_{FF}(k)$.

In the disk representation of the tree $T_{FF}(k)$, we use disks of radii between 1 and α^{k-1}. So, $T_{FF}(k)$ is an α^{k-1}-bounded disk graph. An example of the construction is depicted in Figure 2.

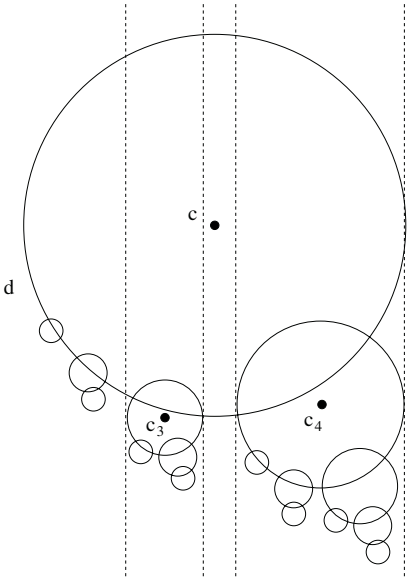


Fig. 2. The disk representation of the tree produced by algorithm ADV on input $k = 5$ and algorithm First-Fit. The dashed lines indicate the boundaries of the horizontal stripes of two disks

Now, consider the forest created by the adversary on input an integer k and some other algorithm A. In this case, some iterations may have renamed some nodes instead of introducing new ones. We construct the disk representation of such a forest by starting with the disk representation of $T_{FF}(k)$. We follow the execution of the adversary on input k and some algorithm A. When, the adversary executes the else statement in an iteration, we remove the disk corresponding to the node $\langle str(j, k-i-1), i\rangle$ (since this node is not introduced) and move horizontally all the disks corresponding to nodes of the subtree of node $\langle s, t\rangle$ so that the disk d corresponding to node $\langle s, t\rangle$ (and no other disk in its subtree) overlaps with the disk corresponding to the parent node of node $\langle str(j, k-i-1), i\rangle$ in $T_{FF}(k)$. From now on, the node to which disk d corresponds has been renamed as $\langle str(j, k-i-1), i\rangle$.

Clearly, this also yields an α^{k-1}-bounded disk graph. The above discussion leads to the following lemma.

Lemma 1. *For any $\alpha > 2$, the forest constructed by the adversary ADV on input an integer $k \geq 3$ and any deterministic online coloring algorithm A is an α^{k-1}-bounded graph.*

Now given a sufficiently large σ, the graph produced by the adversary on input $k = 1 + \lfloor \log_\alpha \sigma \rfloor$ and any deterministic algorithm is a 2-colorable σ-bounded disk graph which the algorithm colors with at least $1 + \lfloor \log_\alpha \sigma \rfloor$ colors. We obtain the following.

Theorem 2. *Any deterministic online algorithm for coloring σ-bounded disk graphs that does not use the disk representation has competitive ratio $\Omega(\log \sigma)$.*

For unit disk graphs, the best known lower bound on the competitiveness of any deterministic algorithm is 2 [6] and holds also for algorithms that use the disk representation. On input a deterministic online algorithm A, the adversary in the proof of [6] constructs a κ-colorable unit disk graph with $\kappa \in \{1, 2, 3\}$, which algorithm A colors with at least 2κ colors. In the following, we improve this lower bound for deterministic online coloring algorithm in unit disk graphs that do not use the disk representation.

Consider a deterministic online coloring algorithm A and the forest produced by adversary ADV on input 5 and algorithm A. Each connected component of the forest produced by ADV is a subtree of the tree $T_{FF}(5)$ produced by ADV on input 5 and algorithm First-Fit. The tree $T_{FF}(5)$ is a unit disk graph as shown in Figure 3.

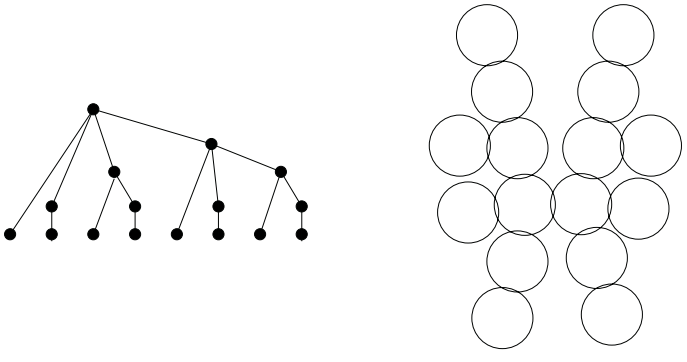


Fig. 3. The tree $T_{FF}(5)$ and its disk representation with unit disks

Hence, the output of the adversary ADV on input 5 and any deterministic algorithm A is a unit disk graph. Since this output is a forest, it is 2-colorable, while the adversary forces A to use at least 5 colors. We obtain the following.

Theorem 3. *Any deterministic online algorithm for coloring unit disk graphs that does not use the disk representation has competitive ratio at least* 2.5.

4 Open Problems

Our results on σ-bounded disk graphs can be extended to other classes of geometric graphs such as intersection graphs of squares and intersection graphs of

rectangles whose height to width ratio is bounded by a constant. It still remains to show whether there exist deterministic online coloring algorithms that use the disk representation and have competitive ratio $o(\log \sigma)$. Our lower bound clearly fails in this case since a very simple online algorithm could use the information for the radii of the disks produced by the adversary ADV and color disks of radius α^i with colors 1 and 2 depending on whether i is even or odd.

Also, it would be interesting to investigate whether randomization helps in improving the known upper bounds and even beating the lower bounds for deterministic algorithms. To our knowledge, randomized online coloring algorithms have not been studied except for very general classes of graphs (e.g., in [16]) where the results are much weaker than those for disk graphs.

References

1. A. Borodin and R. El-Yaniv. Online computation and competitive analysis. *Cambridge University Press*, 1998.
2. I. Caragiannis, A. V. Fishkin, C. Kaklamanis, and E. Papaioannou. Online algorithms for disk graphs. In *Proc. of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS '04)*, LNCS 3153, Springer, pp. 215-226, 2004.
3. B.N. Clark, C.J. Colbourn, and D.S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86, pp. 165-177, 1990.
4. T. Erlebach and J. Fiala. On-line coloring of geometric intersection graphs. *Computational Geometry: Theory and Applications*, 9(1-2), pp. 3-24, 2002.
5. T. Erlebach and J. Fiala. Independence and coloring problems on intersection graphs of disks. *Manuscript*, 2003.
6. J. Fiala, A.V. Fishkin, and F.V. Fomin. Off-line and on-line distance constrained labeling of graphs. In *Proc. of the 9th Annual European Symposium on Algorithms (ESA '01)*, LNCS 2161, pp. 464-475, 2001.
7. A. Gräf. Coloring and recognizing special graph classes. Musikinformatik und Medientechnik Bericht 20/95, Johannes Gutenberg, Universität Mainz, 1995.
8. A. Gräf, M. Stumpf, and G. Weissenfels. On coloring unit disk graphs. *Algorithmica*, 20(3), pp. 277-293, 1998.
9. A. Gyárfás and J. Lehel. On-line and first fit colorings of graphs. *Journal of Graph Theory*, 12(2), pp. 217-227, 1988.
10. D.K. Hale. Frequency assignment: theory and applications. In *Proc. of the IEEE*, 68(12), pp. 1497-1514, 1980.
11. P. Hliněný and J. Kratochvíl. Representing graphs by disks and balls. *Discrete Mathematics*, 229(1-3), pp. 101-124, 2001.
12. S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11, pp. 53-72, 1994.
13. E. Malesińska. Graph theoretical models for frequency assignment problems. *PhD Thesis*, Technical University of Berlin, 1997.
14. M.V. Marathe, H. Breu, H.B. Hunt III, S.S. Ravi, and D.J. Rosenkrantz. Simple hueristics for unit disk graphs. *Networks*, 25, pp. 59-68, 1995.
15. R. Peeters. On coloring j-unit sphere graphs. Technical report, Dept. of Economics, Tilburg University, 1991.
16. S. Vishwanathan. Randomized online coloring of graphs. In *Proc. of the 31st Annual Symposium on Foundations of Computer Science (FOCS '90)*, 1990.

Divide and Conquer Is Almost Optimal for the Bounded-Hop MST Problem on Random Euclidean Instances

(Extended Abstract)

Andrea E.F. Clementi[1], Miriam Di Ianni[1], Angelo Monti[2], Massimo Lauria[2], Gianluca Rossi[1,⋆], and Riccardo Silvestri[2]

[1] Dipartimento di Matematica,
Università degli Studi di Roma"Tor Vergata",
Via della Ricerca Scientifica 1, 00133 Roma - Italy
{clementi, diianni, rossig}@mat.uniroma2.it
[2] Dipartimento di Informatica,
Università degli Studi di Roma "La Sapienza",
Via Salaria 113, 00198 Roma - Italy
lauria@mclink.it, {monti, silver}@di.uniroma1.it

Abstract. The d-DIM h-HOPS MST problem is defined as follows: Given a set S of points in the d-dimensional Euclidean space and $s \in S$, find a minimum-cost spanning tree for S rooted at s with height at most h. We investigate the problem for any constants h and $d > 0$. We prove the first non trivial lower bound on the solution cost *for almost all* Euclidean instances (i.e. the lower-bound holds with high probability). Then we introduce an easy-to-implement, very fast divide and conquer heuristic and we prove that its solution cost matches the lower bound.

1 Introduction

Given a positive integer h, an *h-tree* T is a rooted tree such that the number of hops (edges) in the path from the root to any other node is not greater than h. The *cost* of T, denoted as $\mathsf{cost}(T)$, is the sum of its edge weights. The *Minimum h-hops Spanning Tree* problem (h-HOPS MST) is defined as follows: Given a graph $G(V, E)$ with nonnegative edge weights and a node $s \in V$, find a minimum-cost h-tree rooted at s and spanning G. The h-HOPS MST problem and the related problem in which the constraint is on the tree diameter find applications in several areas: networks [4], distributed system design [22, 7], bit-compression for information retrieval [6].

The efficient construction of a (minimum) spanning tree of a communication network yields good protocols for *broadcast* and *anti-broadcast*[1] operations.

⋆ Supported by the European Union under the IST FET Project CRESCCO.

[1] The anti-broadcast operation is also known in literature as *Accumulation* or *All-to-One* operation.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 89–98, 2005.

The hop restriction limits the maximum number of links or connections in the communication paths between source and destination nodes: It is thus closely related to restricting the maximum delay transmission time of such fundamental communication protocols. The hop restriction finds another relevant application in the context of *reliability*: Assume that, in a communication network, link faults happen with probability p and that all faults occur independently. Then, the probability that a multi-hop transmission fails *exponentially* increases with the number of hops. Summarizing, a *fixed* bound on the maximum number of hops is sometimes a necessary constraint in order to achieve *fast* and *reliable* communication protocols.

For further motivations in studying the h-HOPS MST problem see [5, 11, 15, 24].

The h-HOPS MST problem is NP-hard even when the edge weights of the input graphs form a *metric* and $h = 2$ [1]. Algorithmic research on this issue thus aims to design and analyse efficient approximation algorithms.

Several previous works [5, 11, 24] focused on the 2-dimensional geometric version of the problem (2-DIM h-HOPS MST), i.e., when nodes are points of the Euclidean 2-dimensional space, the graph is complete, and the edge weights are the Euclidean distances. As for the case $h = 2$, polynomial-time, constant-factor approximation algorithms are given in [23, 8, 17, 14, 18] and a PTAS is provided in [3] for 2-DIM h-HOPS MST. We remark that all such approximation algorithms are not fast and/or easy-to-implement and, for $h \geq 2$, neither hardness results nor polynomial-time (exact) algorithms are known for the 2-DIM h-HOPS MST problem. Even more, for $h \geq 3$, no polynomial-time, constant-factor approximation algorithms are known.

Another series of papers have been devoted to evaluate and compare solutions for the 2-DIM h-HOPS MST problem returned by some heuristics on *random* planar instances by performing computer experiments [9, 11, 12, 21, 24]. Almost all such works adopt the *uniform input random model*, i.e., points are chosen independently and uniformly at random from a fixed square of the plane. The motivation on this input model is twofold: On one hand, the uniform distribution is the most suitable choice when *nothing* is known about the *real* input distribution or when the goal is to perform a preliminary study of the heuristic on arbitrary instances. On the other hand, uniform distribution well models important applications in the area of ad-hoc wireless and sensor networks: In such scenarios, once *base* stations are efficiently located, a large set of small wireless (mobile or not) devices are *well-spread* over a geographical region. Clearly, in these networks, efficient and reliable protocols for broadcast and accumulation is a primary goal [10].

We emphasize that no theoretical analysis is currently available on the expected performance of any efficient algorithm for the 2-DIM h-HOPS MST problem.

In the sequel, with the term *random instance*, we mean a finite set of points chosen independently and uniformly at random from a fixed d-dimensional hypercube.

Our first result is a lower bound on the cost of any h-tree spanning a random set of points, i.e., a finite set of points chosen independently and uniformly at random from a fixed d-dimensional hypercube (d-cube).

Theorem 1. *Let $h, d \geq 1$ be constants. Let S be a random set of n points in a d-cube of side length L and let T be any h-tree spanning S. Then, it holds that*

$$\mathsf{cost}(T) = \begin{cases} \Omega\left(L \cdot n^{\frac{1}{h}}\right) & \text{if } d = 1 \\ \Omega\left(L \cdot n^{1-\frac{1}{d}+\frac{d-1}{d^{h+1}-d}}\right) & \text{otherwise} \end{cases} \quad \text{with high probability.}$$

Here and in the sequel the term *with high probability* (in short, *w.h.p.*) means that the event holds with probability at least $1 - e^{-c \cdot n}$, for some constant $c > 0$. So, according to our input model, claiming that a given bound holds w.h.p. is equivalent to claiming that it holds *for almost all* inputs.

We then introduce a simple *Divide et Impera* heuristic h-Party. It makes a partition of the smallest d-cube containing S into *cells*. In each non-empty cell, it selects an *arbitrary* sub-root s' and connects s' to the root s; finally, it solves the non-empty cell sub-instances of the problem with $h - 1$ hops, recursively. Choosing the size of the cells is the critical technical issue: This is solved by means of the lower bound in Theorem 1.

Theorem 2. *Let $h, d \geq 1$ be constants. Let S be a set of n points in a d-cube of side length L and let $s \in S$. For any h-tree T returned by h-Party on input (S, s), it holds that*

$$\mathsf{cost}(T) = \begin{cases} \mathrm{O}\left(L \cdot n^{\frac{1}{h}}\right) & \text{if } d = 1 \\ \mathrm{O}\left(L \cdot n^{1-\frac{1}{d}+\frac{d-1}{d^{h+1}-d}}\right) & \text{otherwise.} \end{cases}$$

Theorems 1 and 2 imply that, for any fixed h, h-Party returns a solution which is, with high probability, a constant factor approximation of the optimum. So, even though this fast algorithm provides no provably-good approximation in the *worst-case*, it works well on *almost-all* Euclidean instances.

h-Party is the first heuristic that works in $O(n)$ time and it can be thus efficiently applied to very large instances. In fact, the heuristic has been implemented and tested on instances of hundreds of thousands points [9].

Notice that, differently from Theorem 1, the bound in Theorem 2 holds for *any* Euclidean instance. It thus follows that random instances are those having the largest cost.

1.1 Related Works

The 2-Dim 2-hops MST problem can be easily reduced to the classic Facility Location Problem on the plane. Indeed, the distance of the root from vertex i can be seen as the cost of opening a facility at vertex i. It thus follows that all the approximation algorithms for the latter problem apply to the 2-Dim 2-hops

MST as well. In particular, the best result is the PTAS given by Arora et al in [3]. The algorithm works also in higher dimensions; however, it is based on a complex dynamic programming technique that makes any implementation very far to be practical. Several polynomial-time approximation algorithms for the METRIC 2-HOPS MST problem have been presented in the literature. Notice that, in [1] Alfandari and Paschos proved that METRIC 2-HOPS MST is MAX SNP-hard and, hence, PTAS cannot be found for this problem unless P = NP. The first constant factor approximation algorithm was given by Shmoys et al in [23], they presented a 3.16 approximation algorithm. After this, a series of constant factor approximation algorithms was published, see [8, 17, 14]. Currently, the best factor is 1.52 due to Mahdian et al [18]. All such algorithms are not practically efficient.

As for the general h-HOPS MST problem, Gouveia [12] and, successively, Gouveia and Requejo [13] provided and experimentally tested exact super-polynomial time algorithms, based on the branch and bound technique. In [2, 16] a polynomial-time $O(\log n)$-approximation algorithm is given, but its time complexity is $n^{O(h)}$. Voss in [24] presented a tabu-search heuristic for the h-HOPS MST problem, but the time complexity is very high when the graph is dense. In [21] heuristics based on Prim algorithm and on Evolutionary techniques have been experimentally tested. Finally, in [9] experimental tests have been performed on greedy heuristics and on h-PARTY.

2 Preliminaries

In the proof of our results we make use of the well-known Hölder inequality. We thus present it in the following convenient forms. Let x_i, $i = 1, \ldots, k$ be a set of k non negative reals and let $p, q \in \mathcal{R}$ such that $p \geq 1$ and $q \leq 1$. Then, it holds that:

$$\sum_{i=1}^{k} x_i^p \geq k \left(\frac{\sum_{i=1}^{k} x_i}{k} \right)^p ; \tag{1}$$

$$\sum_{i=1}^{k} x_i^q \leq k \left(\frac{\sum_{i=1}^{k} x_i}{k} \right)^q . \tag{2}$$

3 The Lower Bound

The next lemma is the first known lower bound on the cost of h-trees for general Euclidean instances.

Lemma 1. *Let $h, d \geq 1$ be constants. Let S be a set of points in a d-dimensional Euclidean space. Consider a partition of the space in d-cubes with the side length of each d-cube being l and let n_l be the number of the d-cubes containing points of S. For any h-tree T spanning S it holds that*

$$\mathsf{cost}(T) = \begin{cases} \Omega\left(l \cdot n_l^{1+\frac{1}{h}}\right) & \text{if } d = 1 \\ \Omega\left(l \cdot n_l^{1+\frac{d-1}{d^{h+1}-d}}\right) & \text{otherwise.} \end{cases}$$

Proof. We equivalently show that $\mathsf{cost}(T) = \Omega\left(L \cdot n^{1+\frac{1}{g(h)}}\right)$ where

$$g(h) = \begin{cases} d & \text{if } h = 1 \\ d \cdot g(h-1) + d & \text{otherwise.} \end{cases}$$

Notice that $1/g(h) = \frac{d-1}{d^{h+1}-d}$ if $d > 1$ and $1/g(h) = \frac{1}{h}$ if $d = 1$.

Let s be the point root of the tree T and consider a d-sphere centered at s and of radius $r = \Theta(l \cdot (n_l)^{\frac{1}{d}})$ such that the number n'_l of non-empty d-cubes outside the sphere is at least $\frac{n_l}{2}$. Finally let B be the set of points in these n'_l d-cubes.

The proof is by induction on the height h of the tree T. If $h = 1$, for each of the n'_l d-cubes, there is an edge in T of length at least r. This implies that

$$\mathsf{cost}(T) \geq r \cdot n'_l = \Omega\left(l \cdot n_l^{1+\frac{1}{d}}\right) = \Omega\left(l \cdot n_l^{1+\frac{1}{g(1)}}\right).$$

Let $h \geq 2$. Let $A = \{a_1, a_2, \ldots, a_{|A|}\}$ be the set of points whose father is at distance at least $\frac{r}{h}$ and let $\beta = 1 - \frac{1}{d} + \frac{1}{g(h)}$. Two cases may arise.

- *Case* $|A| \geq n_l^{\beta}$. Since there are at least $|A|$ edges of length $\frac{r}{h}$, it holds that

$$\mathsf{cost}(T) \geq \frac{r}{h} \cdot |A| = \Omega(l \cdot n_l^{\beta+\frac{1}{d}}) = \Omega\left(l \cdot n_l^{1+\frac{1}{g(h)}}\right).$$

- *Case* $|A| < n_l^{\beta}$. For every point x in B there is a path from x to the root s with at most h hops. Since the distance from x to s is at least r in the path there is at least one edge of length at least $\frac{r}{h}$. Hence we can partition the points in $A \cup B$ into $|A|$ subsets $A_1, A_2, \ldots A_{|A|}$ where a point y is in A_i if a_i is the first point in A in the path from y to s. Notice that the points in the subsets A_i, $1 \leq i \leq |A|$, belong to (edge-)disjoint subtrees $T_1, T_2, \ldots, T_{|A|}$ of T where T_i is an $(h-1)$-tree rooted at a_i. Let $n_{l,i}$ be the number of d-cubes containing the points of T_i, $1 \leq i \leq |A|$. It holds that

$$\begin{aligned} \mathsf{cost}(T) &\geq \textstyle\sum_{i=1}^{|A|} \mathsf{cost}(T_i) \\ &= \Omega\left(\textstyle\sum_{i=1}^{|A|} l \cdot {n_{l,i}}^{1+\frac{1}{g(h-1)}}\right) && \text{by inductive hypothesis} \\ &= \Omega\left(l \cdot |A| \cdot \left(\frac{\sum_{i=1}^{|A|} n_{l,i}}{|A|}\right)^{1+\frac{1}{g(h-1)}}\right) && \text{by the Hölder inequality} \end{aligned}$$

$$
\begin{aligned}
&= \Omega\left(l \cdot |A|^{-\frac{1}{g(h-1)}} \cdot {n_l}^{1+\frac{1}{g(h-1)}}\right) && \text{since } \textstyle\sum_{i=1}^{|A|} n_{l,i} \geq n'_l \geq \frac{n_l}{2} \\
&= \Omega\left(l \cdot n_l^{-\frac{\beta}{g(h-1)}+1+\frac{1}{g(h-1)}}\right) && \text{since } |A| < n_l^{\beta} \\
&= \Omega\left(l \cdot n_l^{1+\frac{g(h)-d}{d\cdot g(h-1)\cdot g(h)}}\right) \\
&= \Omega\left(l \cdot n_l^{1+\frac{d\cdot g(h-1)}{d\cdot g(h-1)\cdot g(h)}}\right) && \text{since } g(h) = d \cdot g(h-1) + d. \\
&= \Omega\left(l \cdot n_l^{1+\frac{1}{g(h)}}\right)
\end{aligned}
$$

The thesis follows. □

By applying the probabilistic method of *bounded differences* [19], we derive the following lower bound

Theorem 1. *Let $h, d \geq 1$. Let S be a random set of n points in a d-cube of side length L and let T be any h-tree spanning S. Then, it holds that*

$$
\mathsf{cost}(T) = \begin{cases} \Omega\left(L \cdot n^{\frac{1}{h}}\right) & \text{if } d = 1 \\ \Omega\left(L \cdot n^{1-\frac{1}{d}+\frac{d-1}{d^{h+1}-d}}\right) & \text{otherwise} \end{cases} \quad \text{with high probability.}
$$

Proof. Let us partition the d-cube into n d-cubes, each of them with side length $l = \frac{L}{n^{\frac{1}{d}}}$. Let n_l be the number of non-empty d-cubes. Lemma 1 implies that

$$
\mathsf{cost}(T) = \begin{cases} \Omega\left(L \cdot n^{-1} \cdot n_l^{1+\frac{1}{h}}\right) & \text{if } d = 1 \\ \Omega\left(L \cdot n^{-\frac{1}{d}} \cdot n_l^{1+\frac{d-1}{d^{h+1}-d}}\right) & \text{otherwise} \end{cases}
$$

The theorem follows by noticing that, by applying the method of *bounded differences* [19], we have that $n_l \geq n/4$, with high probability. □

4 The Divide and Conquer Heuristic

The heuristic h-Party is described in Figure 2 while its solution cost is proved in the following

Theorem 2. *Let $h, d \geq 1$ be constants. Let S be a set of n points in a d-cube of side length L and let $s \in S$. For any h-tree T returned by h-Party on input (S, s), it holds that*

$$
\mathsf{cost}(T) = \begin{cases} \mathrm{O}\left(L \cdot n^{\frac{1}{h}}\right) & \text{if } d = 1 \\ \mathrm{O}\left(L \cdot n^{1-\frac{1}{d}+\frac{d-1}{d^{h+1}-d}}\right) & \text{otherwise.} \end{cases}
$$

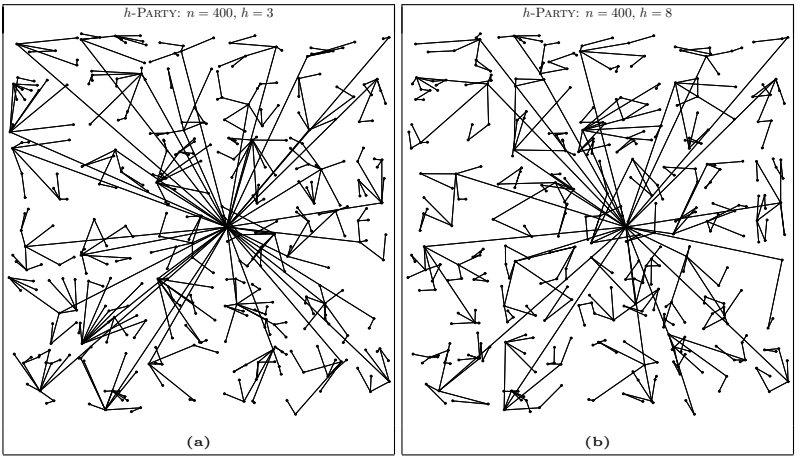


Fig. 1. The trees yielded by the h-Party heuristics on the same random instance with 400 points and $h = 3, 8$

procedure h-Party(S, s)
  if $h = 1$ **then** $T \leftarrow \{\{x, s\} | x \in S - \{s\}\}$;
  else begin
    $T \leftarrow \emptyset$;
    if $d = 1$ **then** $k \leftarrow \left\lfloor |S|^{\frac{1}{h}} \right\rfloor$;
    else $k \leftarrow \left\lfloor |S|^{1-\frac{1}{d}+\frac{d-1}{d^{h+1}-d}} \right\rfloor$;
    Let L be the side length of the smallest d-cube containing all points in S;
    Partition the d-cube into d-cubes of side length $\frac{L}{\left\lceil k^{\frac{1}{d}} \right\rceil}$;
    Let k' be the number of d-cubes and
    let S_i be the points of S in the i-th d-cube, $1 \le i \le k'$;
    for $i \leftarrow 1$ **to** k' **do**
      if $|S_i| \ge 1$ **then begin**
        choose a point s' in S_i;
        $T \leftarrow T \cup \{\{s', s\}\}$;
        if $|S_i| > 1$ **then** $T \leftarrow T \cup (h-1)$-Party$(S_i, s')$;
      end;
  end;
  output T

Fig. 2. Algorithm h-Party

Proof. We equivalently show that $\mathsf{cost}(T) = O\left(L \cdot n^{1-\frac{1}{d}+\frac{1}{g(h)}}\right)$ where

$$g(h) = \begin{cases} d & \text{if } h = 1 \\ d \cdot g(h-1) + d & \text{otherwise} \end{cases}$$

Notice that, as in Lemma 1, $1/g(h) = \frac{d-1}{d^{h+1}-d}$ if $d > 1$ and $1/g(h) = \frac{1}{h}$ if $d = 1$. The proof is by induction on h. If $h = 1$ it is clear that $\mathsf{cost}(T) = O(L \cdot n)$.

For $h \geq 2$, let t be the number of non-empty d-cubes in the d-cube of size length L and $\{q_1, q_2, \ldots q_t\}$ be the set of points selected by the procedure in the t non-empty d-cubes, let T_i be the $(h-1)$-tree rooted in q_i and S_i be the set of points spanned by T_i, $1 \leq i \leq t$. By inductive hypothesis, we get $\mathsf{cost}(T_i) = O\left(\frac{L}{k^{\frac{1}{d}}} \cdot |S_i|^{1-\frac{1}{d}+\frac{1}{g(h-1)}}\right)$. We thus have that

$$
\begin{aligned}
&\mathsf{cost}(T) = \textstyle\sum_{i=1}^{t} d(q_i, s) + \sum_{i=1}^{t} \mathsf{cost}(T_i) \\
&\leq L \cdot t + \textstyle\sum_{i=1}^{t} \mathsf{cost}(T_i) && \text{since } d(q_i, s) \leq L \\
&= O\left(L \cdot t + \textstyle\sum_{i=1}^{t} \frac{L}{k^{\frac{1}{d}}} \cdot |S_i|^{1-\frac{1}{d}+\frac{1}{g(h-1)}}\right) && \text{by inductive hyp.} \\
&= O\left(L \cdot t + \frac{L}{k^{\frac{1}{d}}} \cdot t \cdot \left(\frac{\sum_{i=1}^{t} |S_i|}{t}\right)^{1-\frac{1}{d}+\frac{1}{g(h-1)}}\right) && \text{by the Hölder ineq.} \\
&= O\left(L \cdot t + \frac{L}{k^{\frac{1}{d}}} \cdot t^{\frac{1}{d}-\frac{1}{g(h-1)}} \cdot n^{1-\frac{1}{d}+\frac{1}{g(h-1)}}\right) && \text{since } \textstyle\sum_{i=1}^{t} |S_i| = n \\
&= O\left(L \cdot k + L \cdot k^{-\frac{1}{g(h-1)}} \cdot n^{1-\frac{1}{d}+\frac{1}{g(h-1)}}\right) && \text{since } t \leq k \\
&= O\left(L \cdot n^{1-\frac{1}{d}+\frac{1}{g(h)}} + L \cdot n^{-\frac{1}{g(h-1)}(1-\frac{1}{d}+\frac{1}{g(h)})+1-\frac{1}{d}+\frac{1}{g(h-1)}}\right) \\
&= O\left(L \cdot n^{1-\frac{1}{d}+\frac{1}{g(h)}} + L \cdot n^{1-\frac{1}{d}+\frac{g(h)-d}{d \cdot g(h-1) \cdot g(h)}}\right) \\
&= O\left(L \cdot n^{1-\frac{1}{d}+\frac{1}{g(h)}}\right).
\end{aligned}
$$

where the last step follows since

$$\frac{g(h)-d}{d \cdot g(h-1) \cdot g(h)} = \frac{d \cdot g(h-1)}{d \cdot g(h-1) \cdot g(h)} = \frac{1}{g(h)} \ .$$

□

Finally, it is not hard to verify that, for any $h > 0$, the worst-case time complexity is $O(n)$.

5 Open Problems

It would be interesting to extend our asymptotical analysis to non constant h (e.g. $h = \Omega(\log n)$).

References

1. L. Alfandari and V.T. Paschos. Approximating minimum spanning tree of depth 2. *Intl. Trans. In Op. Res.* 6:607-622, 1999.
2. E. Althaus, S. Funke, S. Har-Peled, J. Koenemann, E. A. Ramos, M. Skutella, Approximation k-hop minimum-spanning trees, *Operations Research Letters*, 33, 115- 120, 2005.

3. S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-medians and related problems. *Proc. 30-th ACM Symposium on Theory of Computing* 106-113, 1998.
4. K. Bala, K. Petropoulos, and T.E. Stern. Multicasting in a Linear Lightwave Network. *Proc. of INFOCOM*, 1350–1358, 1993.
5. A. Balakrishnan and K. Altinkemer. Using a hop-constrained model to generate alternative communication network design. *ORSA Journal of Computing* 4: 147-159, 1992.
6. A. Bookstein and S.T. Klein. Compression of Correlated Bit-Vectors. *Information Systems*, 16(4):110–118, 1996.
7. R. Chou and T. Johnson. *Distributed Operating Systems and Algorithms.* Addison-Wesley. Reading, MA 1997
8. F.A. Chudak. Improved approximation algorithms for uncapacitated facility location problem. *Proc. of the 6-th Conference on Integer Programming and Combinatorial Optimization* 1998.
9. A.E.F. Clementi, M. Di Ianni, A. Monti, G. Rossi, and R. Silvestri, Experimental Analysis of Practically Efficient Algorithms for Bounded-Hop Accumulation in Ad-Hoc Wireless Networks, In *Proc. of the IEEE IPDPS-WMAN*, 2005, to appear.
10. A. Ephremides, G.D. Nguyen, and J.E. Wieselthier. On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. In *Proc. of the 19th INFOCOM*, 585–594, 2000.
11. L. Gouveia. Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers and Operations Research* 22: 959-970, 1995.
12. L. Gouveia. Multicommodity flow models for spanning trees with hop constraints. *European Journal of Operational Research* 95:178-190, 1996.
13. L. Gouveia and C. Requejo. A new relaxation approach for the hop-constrain minimum spanning tree problem. *European Journal of Operational Research* 132:539-552, 2001.
14. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms* 31:228-248, 1999.
15. M. Haenggi. Twelve Reasons not to Route over Many Short Hops, in *Proc. IEEE Vehicular Technology Conference (VTC'04 Fall)*, (Los Angeles, CA), Sept. 2004.
16. G. Kortsarz and D. Peleg. Approximating the weight of shallow Steiner trees. *Discrete Applied Mathematics* 93:265-285, 1999.
17. M.R. Korupolu, C.G. Plaxton and R. Rajaraman. Analysis of a Local Search Heuristic for Facility Location Problems. *Proc. of the 9th annual ACM-SIAM Symposium on Discrete algorithms* 1-10, 1998.
18. M. Mahdian, Y. Ye and J. Zhang, A 1.52-approximation algorithm for the uncapacitated facility location problem. *Proc. of APPROX 2002* LNCS 2462: 229-242, 2002.
19. C.J.H. McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics: Invited Papers at the 12th British Combinatorial Conference.* J. Siemons Ed. London Mathematical Society Lecture Notes Series, 141:148-188, 1989.
20. P. Raghavan and R. Motwani, *Randomized Algorithms.* Cambridge Univ. Press, 1995.
21. G. R. Raidl and B. A. Julstrom, Greedy Heuristics and an Evolutionary Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem. *Proc. of the 2003 ACM Symposium on Applied Computing* 747-752, 2003.
22. K. Raymond. A Tree-Based Algorithm for Distributed Mutual Exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.

23. D.B. Shmoys, E. Tardos and K. Aardal, Approximation algorithms for facility location problems. *Proc. of the 29-th Annual ACM Symposium on Theory of Computing* 265-274, 1997.
24. S. Voss, The steiner tree problem with hop constraint, *Annals of Operations Research* 86:321-345, 1999.

Distributed Exploration of an Unknown Graph

Shantanu Das[1], Paola Flocchini[1], Amiya Nayak[1], and Nicola Santoro[2]

[1] School of Information Technology and Engineering, University of Ottawa, Canada
{shantdas,flocchin,anayak}@site.uottawa.ca
[2] School of Computer Science, Carleton University, Canada
santoro@scs.carleton.ca

Abstract. We consider a group of identical asynchronous agents, initially located at different nodes of an undirected simple graph. The nodes of the graph are unlabeled, and each contains a whiteboard where the visiting agents can write to and read from. The agents, initially, do not know the graph nor its topology. The only a-priori knowledge the agents may have is either the number n of nodes, or the total number k of agents. The goal is for the agents to construct a labelled map of the unknown graph, the same for all agents, so to be in complete agreement with each-other about their environment. This problem, called Labelled Map Construction, is closely related to a variety of other basic problems, including election and rendezvous. We are interested in efficient and generic protocols that can solve the problem, irrespective of the graph topology, where the cost of the algorithm is measured in terms of the total number of moves (or, edge traversals) made by the agents. We present a novel deterministic algorithm that, provided that n and k are co-prime (a necessary condition), constructs a map of the graph, elects a leader among the agents, and provides a unique labelling on the nodes of the graph. Our algorithm uses no more than $O(k\ m)$ edge traversals where m is the number of edges in the graph. Our result improves on the finding by Barriere *et al.* [4] for graphs with sense of direction, extending it to graphs with arbitrary labelling, provided that one of n or k is known.

1 Introduction

1.1 Labelled Map Construction

The problem of exploring and mapping an unknown environment has been extensively studied due to its various applications in different areas. Some examples are navigating a robot through a terrain containing obstacles, finding a path through a maze, or searching a computer network using mobile software agents. The environment to be explored is often modelled as a graph, (or sometimes a digraph) containing one or more mobile entities (we shall call them agents) that can move through the edges of the graph; the goal is to create a map of the graph.

We consider here a distributed version of this problem where several asynchronous agents are initially located at different nodes of a graph. The agents are

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 99–114, 2005.

identical, have the same capabilities and follow the same protocol—in fact they are indistinguishable from one-another. The graph and its topology are initially not known to the agents. At each node of the graph there is a whiteboard that can be accessed by the visiting agents in fair mutual exclusion. The goal for the agents is to construct a labelled map of the unknown graph, the same for all agents, so to be in complete agreement about their environment. We call this problem *labelled map construction* (LMC).

If the nodes of the graph are labelled with unique identifiers and these labels are visible to the agents, the LMC problem can be resolved quite easily: a simple depth-first traversal of the graph (by each agent) would suffice. However, it may be the case that the labels assigned to the nodes of the graph are not visible to the agents. For example, if the agent is a robot traversing a *graph-like* world, it may not have enough sensory capabilities to uniquely identify a node when it reaches it, or if the agent is a software agent traversing a network, it may not have access to the identification field of the nodes in the network. Thus, from the viewpoint of the agents, we can assume the nodes of the graph to be unlabeled (*anonymous*), so that all nodes of same degree look the same to an agent. Clearly, in order to explore such an anonymous graph, the agents need to somehow mark the nodes (by writing on the whiteboards), so that previously visited nodes can be identified on subsequent visits. However, having multiple identical agents creates problems here: marks made by one agent could be indistinguishable from those made by another; different agents might mark in the same way different nodes. Thus, it is not clear whether the agents can successfully map an anonymous graph.

This problem, as we show, is closely related to some others basic problems, like *Agent Election*, *Labelling* and *Rendezvous* in such a way that solving any one of these problems will allow us to solve all the others too. We are interested in the necessary conditions under which LMC can be solved. We are also interested in efficient and generic protocols that can solve the problems, irrespective of the graph topology, where the cost of the algorithm is measured in terms of the total number of moves (or, edge traversals) made by the agents.

1.2 Related Work

Most of the previous work on exploration of unknown graphs has been limited to single agent exploration. Studies on exploration of *labelled* graphs (or digraphs), have emphasized minimizing the cost of exploration in terms of either the number of moves (edge traversals) or, the amount of memory used by the agent (e.g., see [1, 9, 7, 20, 21]).

Exploration of *anonymous* graphs is possible only if the agents are allowed to mark the nodes in some way; except when the graph has no cycles (i.e. the graph is a tree [13, 10]). For exploring arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [5] proposed the method of dropping a pebble on a node to mark it and showed that any strongly connected directed graph can be explored using just one pebble, if the size of the graph is known and using $O(\log \log n)$ pebbles, otherwise. Dudek *et al.* [11] used

a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [6] was to employ two cooperating agents, one of which would stand on a node, thus marking it, while the other explores new edges. The whiteboard model, which we use here, has been earlier used by Fraigniaud and Ilcinkas [14] for exploring directed graphs and by Fraigniaud et al. [13] for exploring trees. In [10, 14] the authors focus on minimizing the amount of memory used by the agents for exploration (they however do not require the agents to construct a map of the graph).

There have been very few results on exploration by more than one agent. As mentioned earlier, a two agent exploration algorithm for directed graphs was given in [6], whereas Fraigniaud et al. [13] showed how k agents can explore a tree. In both these cases however, the agents are co-located (i.e. they start from same node at the same time) and they have distinct identities.

The other problems which have been studied in the similar setting of mobile agents dispersed in a graph are the problems of Rendezvous (i.e. gathering the agents in a single node) and Agent Election (electing a leader among the agents). The research on the *Rendezvous* problem is rather extensive; for a recent account see [2]. However, most of these results are obtained using probabilistic algorithms. Among deterministic solutions to the problem, the investigations of Yu and Yung on synchronous graphs of known topology [23] and of Dessmark *et al.* on synchronous rings and graphs [8] are limited to agents with *distinct* labels. In the context of anonymous agents in an unlabeled network, the only known results are those of Flocchini *et al.* and of Kranakis *et al.* on ring networks using pebbles [12, 19], and those of Barriere *et al.* on graphs with *sense of direction* [4]. In [3], Barriere *et al.* consider solutions to the agent election problem in presence of distinct but incomparable agent labels.

The results obtained in [4] are closely related to our results. In that paper, the authors consider the rendezvous and agent election problem in a setting very similar to our model. However one major difference is that in their model, the edge-labelling on the graph is not arbitrary—rather the labelling provides a *sense of direction* to the agents.

Our work is also related to some of the classical results in the traditional distributed computing model (where the computing entities are stationary processors communicating through message passing). In that model, Gallager, Humblet and Spira [17] gave a distributed algorithm for leader election and spanning tree construction, in *labelled* graphs. Korach, Kutten and Moran [18] showed that the leader election problem is closely related to graph exploration and they proposed the territory acquisition approach for electing a leader in arbitrary (but labelled) graphs. Sakamoto [22] considered anonymous networks and has given an algorithm that builds a spanning forest of the graph under a variety of initial conditions.

1.3 Our Results

We first examine the necessary conditions under which LMC can be solved. At least, the agents need to know the value of n (the number of nodes in the graph)

or k (the number of agents). Furthermore, the value of n and k need to be co-prime to each-other.

We show, that under these conditions, it is indeed possible to solve the *Labelled Map Construction* problem in finite time and efficiently. In fact, we present a protocol that will allow a team of k anonymous agents scattered in an unknown unlabeled graph of n nodes and m links to construct a map of the graph, elect a leader among the agents, and provide a unique labelling on the nodes of the graph. Our algorithm uses no more than $O(km)$ edge traversals.

Our result improves on the earlier result by Barriere *et al.* [4] that, when k and n are co-prime, *Agent Election* is possible if the graph is endowed with *sense of direction*. We show that it is possible to achieve *Agent Election* even in the absence of *sense of direction*, provided that the agents know the value of either n or k. (This is certainly a weaker assumption because the value of n can always be determined by the agents in a graph having *sense of direction*.)

In section 2.2 we formally describe the LMC and other related problems and discuss about their relationship. In section 3, we give an algorithm for collaborative exploration of the graph by multiple agents, that uses only a single bit of whiteboard memory and makes $O(m)$ moves. We then extend this algorithm, in section 4, to construct a spanning tree of the graph, elect a leader among the agents and build a uniquely labelled map of the graph. Finally, we show the correctness of our protocol in section 5. Due to the space constraint, the proofs of some of the lemmas are omitted.

2 Model and Problems

2.1 Model

The environment to be explored by the agents is a simple undirected connected graph $G = (V, E)$ having $n = |V|$ nodes. The labels (if any) on the nodes are invisible to the agents, so that the nodes are anonymous to the agents. However, an agent visiting a node can distinguish among the various edges incident at that node[1]. In other words, the edges incident to a node in the graph are locally labelled with distinct port numbers. However, this labelling is totally arbitrary and there is no coherence between the labels assigned to edges at the various nodes. Without loss of generality, we assume that the links incident at a node u are labelled as $1, 2, 3, \ldots, d(u)$, where $d(u)$ is the degree of that node. Note that each edge $e = (u, v)$ has two labels, one for the link or port at node u and another for the link at node v. We denote the former label as $l_u(e)$ and the later as $l_v(e)$; these two labels are possibly different. The edge labelling of the graph G is specified by $\lambda = \{l_v : v \in V\}$, where for each vertex u, $l_u : \{(u, v) \in E : v \in V\} \rightarrow \{1, 2, 3, \ldots, d(u)\}$ defines the labelling on its incident edges. We denote by Δ, the maximum degree of a node in the graph.

[1] This assumption is necessary because otherwise an agent cannot even explore a simple three-legged-star graph.

There are k agents and each agent starts from a distinct node of the graph, called its *homebase*. The agents have computing and storage capabilities, execute the same protocol, and can move from a node to the neighboring node in G. After moving from u to v, an agent has the label $l_u(u, v)$ of the edge from which it departed, as well as the label $l_v(v, u)$ of the edge from which it arrived. Whenever an agent reaches a node, it can communicate directly with the other agents present in that node.

The agents are *anonymous* in the sense that they do not have distinct names or labels. They execute a protocol (the same for all agents) that specifies the computational and navigational steps. They are *asynchronous*, in the sense that every action they perform (computing, moving, etc.) takes a finite but otherwise unpredictable amount of time.

An agent can communicate with other agents by leaving a written message at some node which can be read by any agent visiting that node. Thus, in our model, each node of the network is provided with a *whiteboard*, i.e., a local storage where agents can write and read (and erase) information; access to a whiteboard is done in mutual exclusion. The whiteboards are also used for marking the nodes. Initially, the homebases of the agents are marked[2]. The amount of whiteboard memory available at a node, is limited; $O(\log n)$ bits suffice for our algorithms.

Initially the agents do not know the graph or its topology. The only a-priori knowledge the agents may have is either the size of the graph, n, or the total number of agents present, k.

2.2 Problems and Constraints

The *Labelled Map Construction* problem consists in having the agents construct the same map of the graph, where both edges and nodes are labelled; the edges labels are same as those of the graph, while there is no a-priori restriction on the node labels except that each node must have a unique label. We assume that the amount of local memory available with an agent is enough to store such a map.

Formally, the LMC problem can be stated as follows. An instance of the problem consists of a graph $G(V, E)$ with an edge-labelling λ defined on G, and a placement function $p : V \to \{0, 1\}$ which defines the initial locations of the $k = |\{v \in V : p(v) = 1\}|$ agents. A given instance (G, λ, p) of the LMC problem is said to have been solved by a distributed algorithm $\mathcal{A}$, if on executing the algorithm $\mathcal{A}$, each agent obtains a node-labelled, edge-labelled map of the graph, (with the position of the agent marked in it) such that the label assigned to any particular node is the same in all the maps.

In the rest of this section, we look at the relationship between the LMC problem and other related problems for multiple agents dispersed in an unknown environment. For each of these, the problem instance is defined in the same manner as for the LMC problem.

[2] Since both nodes and agents are *anonymous* this marker denotes that the node is the homebase of some agent, but cannot be used to break symmetry.

- The *Labelling* problem : The problem of assigning unique labels to the nodes of an unlabeled graph. A given instance (G, λ, p) of the *Labelling* problem is said to have been solved when the whiteboard of each node is marked with a label and no two nodes have the same label.
- The *Agent Election* problem(AEP) : The problem of electing a leader among the agents. A given instance (G, λ, p) of the AEP problem is said to have been solved when exactly one of the k agents reaches the state 'LEADER' and all other agents reach the state 'FOLLOWER'.
- The *Rendezvous*(RV) problem : The problem of gathering all the agents together in one node. A given instance (G, λ, p) of the Rendezvous problem is said to have been solved when all the k agents are located in a single node of G.
- The *Spanning Tree Construction*(SPT) problem : The problem of constructing a spanning tree of the graph. A given instance (G, λ, p) of SPT problem is said to have been solved if each edge of the graph G is marked as either a Tree-edge or Non-Tree edge, such that the set of Tree-edges, T represents a spanning tree of the graph G (i.e. (V, T) is a tree).

For any of the above problems, an instance of the problem is said to be solvable if there exists a deterministic algorithm $\mathcal{A}$ such that every execution[3] of the algorithm $\mathcal{A}$ solves the particular instance of the problem.

Theorem 1. *The LMC problem is solvable for the instance* (G, λ, p) *if and only if the AEP problem is solvable for the same instance.*

Proof. $LMC => AEP$: Once the LMC problem has been solved, agent election can be done without making any extra moves. When each agent has a uniquely labelled map of the graph, the agent whose homebase has the smallest label in the map, (among all the homebase nodes), changes to 'LEADER' state and all the other agents change their state to 'FOLLOWER'.
$AEP => LMC$: Once a leader agent is elected, this agent can explore the graph, marking the nodes with unique labels, while the other agents remain stationary in their homebases. Thus the leader agent can construct the map and then it can visit the homebase of each agent to communicate the map to the other agents. Note that only $2(m + n)$ extra moves are required in this case.

It is interesting to note that solving the *Rendezvous* problem is also equivalent to solving the agent election problem, in the presence of whiteboards. The mutual exclusion property of the whiteboards allow us to break the symmetry among the agents and elect a leader, once the agents rendezvous at a single node. On the other hand, solving the LMC problem solves both *Labelling* and *Rendezvous*. Once the agents have a labelled map they can mark the whiteboard of the nodes with the respective labels. The agents can then move to the node having the smallest label, thus achieving *Rendezvous*. Both these tasks could be done in

[3] Recall that we are considering an asynchronous system and the agents can start execution at any arbitrary time.

$O(k.n)$ moves. Finally, note that solving the *Labelling* problem helps us to solve the LMC problem. Once the graph is labelled, each agent can execute a depth-first traversal of the labelled graph, to obtain a uniquely labelled map of the graph. During the depth-first traversal, each agent would make $2m$ moves, for total of $2k.m$ moves.

Thus the problems of *Labelling*, *Rendezvous*, *Agent Election* and *Labelled Map Construction* are computationally equivalent in our model. However the SPT problem is not equivalent to these four problems, in general.

The relationship among these problems can be used to determine the conditions for solvability of the LMC problem, based on previous results for the leader election problem.

Lemma 1. *The LMC problem is not solvable if the agents know neither the value of n (the size of the graph) nor k (the number of agents present).*

Lemma 2. *For k agents in a graph of size n, the LMC problem is not solvable, in general, if $\gcd(n, k) > 1$ i.e. if n and k are not co-prime.*

Notice that when n and k are not co-prime, it is possible that the agents are initially placed in exactly symmetrical positions with respect to each other (provided that the graph itself is symmetrical; e.g. a ring), such that, no deterministic algorithm can break the symmetry among the agents and achieve leader election. Also note that Lemma 1 holds even when $\gcd(n, k) = 1$.

For the rest of this paper, we shall assume that $\gcd(n, k) = 1$ and the agents have prior knowledge of the value of at least one of n and k. Under these conditions, we show that it is possible to solve the *Labelled Map Construction* problem, using just $O(\log n)$ node memory and making $O(k.m)$ moves.

3 Distributed Traversal

As a preliminary step in our solution protocol, we will have the agents perform an initial cooperative exploration of the graph. We want the agents to explore the graph collectively, in such a way that the total number of edge traversals is minimized. Each agent can traverse an area around its homebase, while avoiding the parts being explored by the other agents. During the exploration, the agent needs to remember the path to its homebase, so that it does not get lost. Each agent stores in its memory the sequence of labels (in order) of edges traversed by it, starting from the homebase. We call this sequence of labels as the *Exploration Path*. When the edge $e = (u, v)$ is traversed by the agent from u to v, then the label $l_v(e)$ is appended to the *Path*. This enables the agent to return back to the previously visited node (i.e. u) whenever it wants to. When it does so, the agent is said to have backtracked the edge e and the label $l_v(e)$ is deleted from the *Path*. So, at all times during the traversal, the *Path* contains the sequence of labels of the links that an agent has to traverse (in reverse order) to return to its homebase from the current node.

Each agent on wake-up, starts traversing the graph, from the homebase and it marks the visited nodes if they are previously unmarked[4]. The agent also builds a partial *Map* of the territory that it marks. The edges are marked as 'T' or 'NT', where 'T' edges belong to the territory and are included in the Map, whereas, 'NT' edges are not included in the Map. The algorithm executed by each agent is the following:

Algorithm EXPLORE

1. Set *Path* to empty ;
   Initialize the *Map* as single-node graph consisting of the homebase;
2. While there is another unexplored edge e at the current node u,
   mark link $l_u(e)$ as 'T' and then traverse e to reach node v;
   If v is already marked,
   return back to u and re-mark the link $l_u(e)$ as 'NT';
   Otherwise
   mark the link $l_v(e)$ as 'T', and mark v as explored;
   Add link $l_v(e)$ to *Path*;
   Add the edge e to *Map*;
3. When there are no more unexplored edges at the current node,
   If Path is not empty then,
   remove the last link from Path, traverse that link and repeat Step 2;
   Otherwise, Stop and return the *Map*;

We make the following observations about the effects of this algorithm.

Lemma 3. *During the execution of algorithm EXPLORE,*

(a) *If an agent marks a node, it eventually traverses each edge incident to it.*
(b) *Every node in the graph is marked by exactly one agent.*
(c) *If two nodes are marked by the same agent then there exists a consecutive sequence (i.e. a path) of 'T' edges joining them. On the other hand, if two nodes are marked by two different agents, then every path joining them would contain at least one 'NT' edge.*
(d) *There is no cycle consisting of only 'T' edges.*

Lemma 4. *The total number of edge traversals made by the agents in executing algorithm EXPLORE, is at most* $4.m$*, irrespective of the number of agents.*

Proof. During the algorithm EXPLORE, each 'T' edge is traversed twice—once in the forward direction and once while backtracking. Each 'NT' edge is traversed four times, twice from each side. As there are $(n-k)$ 'T' edges, the total number of moves (i.e. edge traversals) made by the agents is $(4m - 2n + 2k)$.

[4] Recall that the homebases are already marked.

Note that for this algorithm, we do not require much memory at the whiteboards of the nodes. In fact one bit per whiteboard is sufficient—for marking the node as explored.

When an agent A finishes executing algorithm EXPLORE, A would have obtained a map of the territory marked by it. This territory is a tree consisting of all the nodes marked by it and the 'T' edges connecting these nodes; the edges leaving the territory are marked 'NT'. The territories marked by different agents are all disjoint, and together they span the whole graph. So, the distributed traversal of the graph by multiple agents creates a spanning forest of the graph. Two adjacent trees in the forest would be connected through one or more 'NT' edges, each joining a node of one tree to that of another.

Notice that some of the 'NT' edges could possibly be connecting nodes of the same tree (i.e. they are *back edges*) while the other 'NT' edges connect nodes from two distinct trees. The agents cannot, in general, distinguish between these two types of 'NT' edges and from the point of view of an agent, all the 'NT' edges, it encounters, seem to be going out of its territory. Thus, the maps constructed by algorithm EXPLORE provide an incomplete view of the graph to the agents and constructing the complete map of the graph does not simply involve joining these partial maps together. In the next section we show how we can merge these maps and also achieve leader election among the agents.

4 Merging the Maps: Spanning Tree Construction

To obtain the map of the whole graph, the maps constructed by the agents after the execution of algorithm EXPLORE, need to be merged somehow. The task of merging together the maps (i.e. the territories) of the agents, is complicated by the fact that the maps constructed by two agents may look exactly similar. Also there may be cyclic 'NT' edges connecting two nodes of the same tree. To avoid the cyclic edges, we would first construct a spanning tree of the graph by joining the trees marked by different agents.

In this section, we show how the agents can construct a spanning tree of the graph and then finally use it to obtain a complete map of the graph. Here, the reader may recall the well-known distributed algorithm for minimum spanning tree construction(MST) given by Gallager, Humblet and Spira [17], where the spanning tree is constructed by repeatedly joining adjacent trees using the unique edge of minimum weight connecting them. Such an approach is unfortunately not applicable in our setting, since neither the edges nor the nodes have unique labels, making it impossible for the agents to agree on a unique edge for joining two trees[5].

[5] Also note that our model is drastically different from that of MST, where the computational entities were immobile (i.e. connected to the nodes) and they communicated by message passing.

Thus, in our setting, we need a much more complicated protocol for merging the maps and building the spanning tree. Such a distributed algorithm, MERGE-TREE, is described in the following.

This new algorithm uses the algorithm *EXPLORE* as a procedure. The algorithm proceeds in phases, where in each phase, some agents become passive i.e. they stop participating in the algorithm. Agents communicate by writing certain symbols on the whiteboards. Two special symbols would be used which we call the 'ADD-ME' symbol and the 'DEFEATED' symbol. An agent can be in one of three states: *Active*, *Defeated* or *Passive*. Each agent is *active* at the time it starts the algorithm, but it may become *defeated* and subsequently *passive*, during some phase of the algorithm. When an agent becomes *passive* during a phase, it keeps waiting at its current location till the end of the algorithm. At the time an agent starts the algorithm, it knows the value of either n or k.

Algorithm *MERGE-TREE*

Phase 0 : Each agent on startup executes procedure EXPLORE and when it finishes, it has a map of the territory marked by it and also a count of the number of nodes marked. Each agent maintains a *Token* which is of the form (Nc, Ac, Ph) where Nc (Node-Count) is the count of the number of nodes marked by it, Ac (Agent-Count) is the number of agents in its territory (initially set to 1) and Ph is the phase number which is also initially set to 1. Now the agent can begin the first phase.

In phase $i \geq 1$, an agent A (if *active*), executes the following steps:

STEP 1 – 'WRITE-TOKEN' : Agent A does a depth-first traversal of its territory using the map; recall that a territory is a tree. During the traversal the agent writes its Token on the whiteboard[6] of each node in its tree.

STEP 2 – 'COMPARE TOKEN' : During this step, the agent compares its Token with the Tokens in adjacent trees. Agent A starts a depth-first traversal of its territory. During the traversal, whenever it finds an 'NT' edge $e = (u, v)$ incident to some node u in its territory, it traverses the edge e to reach the other end v, compares its Token with the Token at v, and takes an appropriate action, before returning back to u. If it does not find any Token at node v (or, finds a Token from the previous phase $i-1$), it waits till the Token for phase i is written at v. On the other hand, if it finds a Token from phase $i+1$ at node v, it ignores that Token, goes back to u and continues with the traversal.

Two Tokens from the i-th phase, $T_1 = (Nc_1, Ac_1, i)$ and $T_2 = (Nc_2, Ac_2, i)$, are compared as follows. Token T_1 is said to be larger than Token T_2 if either $Nc_1 > Nc_2$, or $Nc_1 = Nc_2$ and $Ac_1 > Ac_2$. The two tokens are said to be equal if both $Nc_1 = Nc_2$ and $Ac_1 = Ac_2$. Otherwise, Token T_1 is smaller than Token T_2.

After the comparison of Tokens, the agent takes one of the following actions:

[6] Any previously written Token or symbol is deleted from the whiteboard.

- If the Token at the other side is larger, it writes a 'ADD-ME' symbol on the whiteboard of node v and returns to node u. It remembers[7] the node u, (as the *terminal* node) and the edge e (as the *bridge* edge). It then does a complete traversal of its territory writing 'DEFEATED' symbols on each node in its territory. It now becomes *defeated*. (The actions taken by a *defeated* agent are described below.)
- If the Token at the other side is equal to its own Token, it ignores the Token, returns to its own tree and continues with its traversal.
- If the Token at the other side is smaller, it waits at node v till it finds a 'DEFEATED' symbol. On finding a 'DEFEATED' symbol, it goes back to u and continues with the traversal.

If agent A becomes *defeated* then it takes the following actions. It continues with the traversal and Token comparisons — whenever it finds a Token which is smaller or equal to its Token, it takes the same action as an *active* agent; but, when it finds a Token that is larger than its Token, it ignores it. (So, a *defeated* agent never writes any 'ADD-ME' symbol.) After completing the traversal, the *defeated* agent A returns to the *terminal* node u and marks the *bridge* edge e as a 'T' edge. It then traverses the edge e to reach the other end, say v. It adds the edge e to its map and designates the vertex corresponding to node v, as the *junction point*, in the map. At this stage, the agent A becomes *passive* and does not participate in the algorithm anymore.

During the traversal, whenever an *active* (or *defeated*) agent A finds an 'ADD-ME' symbol at some node w in its tree, it takes the following action. It deletes the 'ADD-ME' symbol and waits at node w till the agent B (which had written the message), returns back to w. Agent A then acquires all the information available in agent B's memory, including B's Token, its map and all other Tokens and maps acquired earlier by agent B. Agent A also remembers the vertex corresponding to node w, as the location where it acquired this new information. (This vertex is called the *acquisition point*.)

STEP 3 – 'UPDATE TOKEN' : If the agent A completes the second step without becoming *passive*, it extends its territory and updates its Token, before starting the next phase. The agent adds together the Node-count and Agent-count values respectively, from all the acquired Tokens, including its own Token, to get the new values of Node-count Nc, and Agent-count Ac. The new phase number is obtained by incrementing Ph by one. The agent also constructs a new map by merging the acquired maps with its own map. Note that the agent has the information about how to merge the maps[8]. (While merging the maps, the agent may have to relabel some of the vertices of the maps, to ensure unique labelling of the vertices.) The resulting map constructed by the agent defines its new territory.

[7] The agent remembers a node by marking in its map.
[8] The maps are disjoint except for the joining vertex.

On updating the Token, if the agent finds that the new node-count is equal to n (or the agent-count is equal to k), then it reaches the termination condition. Otherwise, it proceeds with the next phase[9].

When an agent A reaches the termination condition, it becomes the leader agent; at this stage, it has a spanning tree of the whole graph. Finally, it executes the following procedure:

Procedure COMPLETE-MAP

1. The leader agent executes a depth-first traversal of the spanning tree, writing node labels on the appropriate whiteboards.
2. The leader agent traverses the graph, adding the non-tree edges to the map.
3. The leader agent traverses the spanning tree to communicate the full map to all the agents.

5 Analysis of the Algorithm

In this section, we show the correctness of our algorithm and analyze its complexity. We use the following notations. G_{iA} denotes the subgraph of G that corresponds to the territory of agent A at the time when it reaches the end of phase i. If A becomes *passive* in phase i, then $G_{iA} = \phi$. We denote by Γ_i the set of all agents which start phase i, in *active* state. We say that the algorithm reaches phase i, if there is at least one agent that starts phase i.

Whenever an agent A becomes *defeated* on comparing its Token with the Token of an agent B, during phase i, we say that agent A was defeated by agent B in phase i. In that case we know that B was *active* at the start of phase i and B's Token in phase i was larger than A's Token, in phase i.

The following facts imply that there is no deadlock in the algorithm MERGE-TREE.

Lemma 5. (a) *An (*active*) agent that starts phase i, either completes the phase or becomes* passive *during the phase.*

(b) *At the end of every phase i reached by the algorithm MERGE-TREE, there is always at least one* active *agent.*

Proof. Part(a): We show that there cannot be any cyclic waiting among the agents. Suppose, for the sake of contradiction, that there is a group of agents $A_1, A_2, \ldots, A_t$ such that for each $1 \leq j \leq t-1$, A_j waits for A_{j+1}, and A_t waits for A_1. We represent these agents as vertices of a graph and we draw directed (colored) arcs to denote who waits for whom. (The color of the edge denotes the type of waiting.) There are three situations when an agent A, in phase $i \geq 1$ has to wait at a node v for some agent B:

[9] After k phases, if the agent has neither reached the termination condition nor has become passive, then it may terminate with failure notification.

1. Agent A found no Token, or a Token from phase $(i-1)$ at node v and it is waiting for the Token for phase i to be written, by agent B. [Denoted by *Blue* arc.]
2. Agent A is waiting at node v, after finding an 'ADD-ME' symbol written by B. [Denoted by *Yellow* arc.]
3. Agent A found agent B's Token (at node v) to be smaller than its own Token. This indicates A is waiting for agent B to write a 'DEFEATED' symbol at v. [Denoted by *Red* arc.]

Note that in first case above, agent B is either in a lower phase than A, or B is yet to complete step-1 of phase i. But in the other two cases, both A and B have to be in step-2 of the same phase i and also B would have a smaller Token than A in that phase. Also note that an agent can be waiting only if it is in step-2 (i.e. the COMPARE-TOKEN step) of some phase.

So, each $A_j, 1 \leq i \leq t$, is in step-2 of some phase and for any $1 \leq j, k \leq t$,

- there is a blue arc from A_j to A_k if and only if A_k is in smaller phase than A_j.
- there is a red or yellow arc from A_j to A_k if and only if A_k is in the same phase as A_j but has a smaller Token than A_j.

Let us first consider the case when at least one of the arcs in the cycle is blue. Let A_j be the first node with a blue arc in the cycle (connecting A_j to A_{j+1}). Let A_j be in phase i. By definition of red and yellow arcs, we then know that any A_k with $k < j$ are in the same phase i. If $j = t$, we immediately have a contradiction because, by definition of blue arc, A_1 would have to be in a phase smaller than i. Let us then assume that $2 \leq j \leq k-1$. Also in this case we have a contradiction because, by definition of blue arc, $A_{j+1}, A_{j+2}, \ldots, A_t$ must be in a smaller phase than phase i. So, there can neither be a blue nor a red (or yellow) arc from A_t to A_1. Thus, we cannot have a cycle containing any blue arc.

We now consider the case when the cycle is composed by yellow and red arcs only. In this case, all the agents in the cycle would be in the same phase i, and each agent would have a smaller Token than the agent on its left — which is not possible!

So, we conclude that there can not be any cyclic waiting among the agents. Each agent in phase i, either reaches the end of the phase or becomes *passive*.

Part(b): Note that an agent A can be defeated by an agent B during phase i, only if B's Token in phase i is larger than A's Token, in phase i. So, an agent A having the largest Token in phase i cannot be defeated in phase i. Thus, agent A remains *active* at the end of phase i.

We will now show that the algorithm MERGE-TREE terminates in finite time, and on termination, there is exactly one leader agent and the map constructed by the leader agent is a spanning tree of the graph G.

Lemma 6. *The following holds for any phase i that is reached by the algorithm:*

1. *For each $A \in \Gamma_i$, G_{iA} is a tree.*
2. *For any $A, B \in \Gamma_i$, if A and B are distinct, then $G_{iA} \cap G_{iB} = \phi$.*
3. *$H_i = \bigcup_{A \in \Gamma_i} G_{iA}$, is a subgraph of G having the same vertex set as G.*

For an agent $A \in \Gamma_i$, we define $NodeCount(G_{iA})$ to be the number of nodes in G_{iA}. Note that, this is equal to the Nc part in the Token of agent A for phase $i+1$. Similarly, $AgentCount(G_{iA})$ is defined to be the number of nodes in G_{iA} that are agent homebases. This is equal to the Ac part in the Token of agent A in phase $i+1$. We have the following corollary as a consequence of the above lemma:

Corollary 1. *For any phase i reached by the algorithm, we have*

$$\sum_{A \in \Gamma_i} NodeCount(G_{iA}) = n \quad and \quad \sum_{A \in \Gamma_i} AgentCount(G_{iA}) = k$$

Two agents A and B are said to be neighbors in phase i, if G contains an edge $e = (u, v)$ such that G_{iA} contains the vertex u and G_{iB} contains the vertex v. Note that the edge e is not included in either G_{iA} or G_{iB} (as $G_{iA} \cap G_{iB} = \phi$), so e would remain to be marked as an 'NT' edge at the end of phase i.

Lemma 7. *If* $\gcd(n, k) = 1$ *then, for any phase $i \geq 1$ with $|\Gamma_i| \geq 2$, at least one agent $B \in \Gamma_i$, becomes* passive *during phase i.*

Thus, in each phase, at least one of the *active* agents, becomes *passive*, until in some phase i, there is only a single *active* agent left. The territory of this agent A, would be the tree G_{iA} containing all the nodes of G (due to Lemma 6), and the node-count and agent-count of A would equal n and k respectively. Thus, agent A would reach termination condition and the algorithm would terminate. Note that the termination condition can be reached only if there is a single *active* agent. The algorithm would always terminate within k phases, because in each phase of the algorithm, some agent would become *passive*. So, we have the following theorems:

Theorem 2. *If* $\gcd(n, k) = 1$*, then algorithm MERGE-TREE terminates after at most $i < k$ phases, with a single agent A reaching the termination condition; when this happens, G_{iA} represents a spanning tree of G.*

Theorem 3. *After executing the procedure COMPLETE-MAP, every agent has a uniquely labelled map of the graph.*

This proves the correctness of our algorithm.

Theorem 4. *The number of edge traversals made by the agents during algorithm MERGE-TREE is in $O(k.m)$ where m in the number of edges in the graph G.*

Proof. During procedure EXPLORE, the agents perform at most $4m$ moves. During each phase of the algorithm MERGE-TREE, each 'T' edge is traversed twice in step-1 and twice in step-2, during the depth-first traversals; Each NT edge is traversed at most four times in step-2. This accounts for $4m$ moves per phase and thus a total of $O(k.m)$ moves. Other than that each defeated agent does one extra traversal of its tree to write 'DEFEATED' messages. These extra moves would account for at most $O(k.n)$ edge traversals. Finally, the procedure COMPLETE-MAP takes $O(m)$ edge-traversals.

As for the memory requirement, observe that $O(\log n)$ bits suffice for a whiteboard.

Acknowledgements

This work has been partially supported by NSERC (National Sciences and Engineering Research Council of Canada) and by Tecsis Corporation.

References

1. S. Albers and M. Henzinger, "Exploring unknown environments", In Proc. 29th Annual ACM Symp. Theory Comput., 416–425, 1997.
2. S. Alpern and S. Gal, *The Theory of Search Games and Rendezvous*, Kluwer, 2003.
3. L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, "Can we elect if we cannot compare?", In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures* (SPAA'03), 200–209, 2003.
4. L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, "Election and rendezvous in fully anonymous systems with sense of direction", In *Proc. 10th Coll. on Structural Information and Communication complexity* (SIROCCO'03), 17–32, 2003.
5. M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan, "The power of a pebble: Exploring and mapping directed graphs", In *Proc. 30th ACM Symp. on Theory of Computing* (STOC'98), 269–287, 1998.
6. M. Bender and D. K. Slonim, "The power of team exploration: two robots can learn unlabeled directed graphs", In *Proc. 35th Symp. on Foundations of Computer Science* (FOCS'94), 75–85, 1994.
7. X. Deng and C. H. Papadimitriou, "Exploring an unknown graph". *J. of Graph Theory* 32(3), 265–297, 1999.
8. A. Dessmark, P. Fraigniaud, and A. Pelc, "Deterministic rendezvous in graphs", In *Proc. 11th European Symposium on Algorithms (ESA'03)*, 184–195, 2003.
9. A. Dessmark and A. Pelc, "Optimal graph exploration without good maps", In *Proc. 10th European Symposium on Algorithms (ESA'02)*, 374–386, 2002.
10. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc, "Tree exploration with little memory", *Journal of Algorithms*, 51:38–63, 2004.
11. G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction", *Transactions on Robotics and Automation*, 7(6):859–865, 1991.
12. P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk, "Multiple mobile agent rendezvous in a ring". Proc. *6th Latin American Theoretical Informatics Symp.* (LATIN'04), 599–608, 2004.

13. P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc, "Collective tree exploration", In *6th Latin American Theoretical Informatics Symp.* (LATIN'04), 141–151, 2004.
14. P. Fraigniaud and D. Ilcinkas, "Digraph exploration with little memory", Proc. *21st Symp. on Theoretical Aspects of Computer Science* (STACS'04), Montpellier, 246–257, 2004.
15. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg, "Graph exploration by a finite automaton", In *29th Symposium on Mathematical Foundations of Computer Science* (MFCS), 451–462, 2004.
16. P. Fraigniaud, A. Pelc, D. Peleg, and S. Perennes, "Assigning labels in unknown anonymous networks with a leader", *Distributed Computing* 14(3), 163–183, 2001.
17. R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees", *ACM Transactions on Programming Languages and Systems* 5(1), 66–77, Jan. 1983.
18. E. Korach, S. Kutten, S. Moran, "A Modular Technique for the Design of Efficient Distributed Leader Finding Algorithms", *ACM Transactions on Programming Languages and Systems* 12(1), 84–101, 1990.
19. E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk, "Mobile agent rendezvous in a ring", In *Int. Conf. on Distibuted Computing Systems* (ICDCS 03), 592–599, 2003.
20. P. Panaite and A. Pelc, "Exploring unknown undirected graphs", Proc. *9th ACM-SIAM Symp. on Discrete Algorithms* (SODA'98), 316–322, 1998.
21. P. Panaite and A. Pelc, "Impact of topographic information on graph exploration efficiency", Networks, 36 (2000), 96–103, 2000.
22. N. Sakamoto, "Comparison of initial conditions for distributed algorithms on anonymous networks", Proc. *18th ACM Symposium on Principles of Distributed Computing* (PODC'99), 173–179, 1999.
23. X. Yu and M. Yung, "Agent rendezvous: A dynamic symmetry-breaking problem", In *Int. Coll. on Automata Languages and Programming* (ICALP'96), 610–621, 1996.

Two Absolute Bounds for Distributed Bit Complexity*

Yefim Dinitz[1] and Noam Solomon[2]

[1] Dept. of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel
[2] Dept. of Mathematics, there
{dinitz, noams}@cs.bgu.ac.il

Abstract. The concept of distributed communication bit complexity was introduced by Dinitz, Rajsbaum, and Moran. They studied bit complexity of Consensus and Leader Election, arriving at more or less exact bounds. This paper answers two questions on Leader Election, which remained open. The first is to close the gap between the known upper and lower bounds, for electing a leader by two linked processors. The second is whether the known algorithm, sending $1.5n$ bits while electing a leader in a chain of even length n, is optimal, in the case when n is known to the processors. For both problems, absolutely exact bounds are found. Moreover, the lower bound proofs show that there is no optimal algorithm other than the suggested one(s).

1 Introduction and Model Definition

The concept of *distributed communication bit complexity* was introduced by Dinitz, Rajsbaum, and Moran [3]; it generalizes the known communication complexity measure (see e.g. [6, 2]) to the distributed computing setting. They showed that message complexity is unable to distinguish between complexities of solving Consensus, Leader Election, and Maximal Id Finding (henceforth, denoted `Consensus`, `Leader`, and `MaxF`) in chains and rings, which contradicts intuition. In contrast, the bit complexity bounds proven there distinguish them successfully, which justifies importance of the suggested new complexity measure. In [3] (see its full version in [4]) and the sibling paper [5], several more or less tight pairs of upper and lower bounds for `Consensus`, `Leader`, and `MaxF` in chains, trees, and rings were found. This paper studies two questions on `Leader` in chains, which remained open. The first is to close the gap between known upper and lower bounds, for electing a leader by two linked processors. The second is whether the algorithm [5], sending $1.5n$ bits while electing a leader in a chain of even length n, is optimal, in the case when n is known to the processors. For both cases, absolutely exact bounds are found: For the former question, the recursive formula and explicit expression are given, while for the latter one, optimality of the known algorithm is confirmed. Moreover, in both cases, the lower bound

* Partially supported by the Lynn and William Frankel Center for Computer Science.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 115–126, 2005.

proofs show that there is no optimal algorithm other than the suggested one(s). We hope that our techniques would shed a new light on some fine aspects of distributed computation.

The model we consider consists of a failure-free, asynchronous message passing distributed system, with arbitrary but finite link delays and negligible local computation times, without shared memory; it is a standard one, for more detail see e.g. [1, 7]. The network topology concerned is a chain, with n links and $n+1$ processors. We assume that the only input is distinct ids at terminals, and that all processors are identical, in the sense that they all run the same (deterministic) algorithm, or program. However, the processors may behave differently, because the program of a processor has access to its id, if any, and to its number of incident links. The ids are taken from the set $Z_M = \{0, 1, \ldots, M-1\}$, $M \geq 2$. We say, for short, that two ids are *paired* if they are given to the terminals. For a task T, $BitC(T)$ is the number of bits sent needed to solve T, in the worst case.

The `Leader` task requires that each processor should output (decide on) a binary value "leader"/"non-leader", so that there will be exactly one leader. Besides, it is required that any non-leader should learn which of its incident links is in the direction to the leader. The `MaxF` version requires that the terminal with the maximal id must be chosen to be leader; clearly, $BitC(\texttt{MaxF}) \geq BitC(\texttt{Leader})$.

Initially, all processors are *asleep* and in the same initial state, except for the id and number of incident links information. When a processor wakes up, spontaneously, or when it receives a message on an incident link, it is *activated*. Then, according to the (common) processor algorithm and its own local information, it sends messages on its incident links (it may send zero or more messages on each link), changes its state, may decide "leader"/"non-leader", and enters the waiting state; all this is done immediately. In this paper, following [3, 5], when defining an algorithm, we assume that each message consists of a single bit. In our lower bound proofs, we assume more generally that a processor may send several bits at the same time, but it gets information, sent to it, bit by bit.

There may be different executions beginning from the same initial state. The "worst case", for an algorithm, is the maximal number of bits sent, over all id pairs and all possible executions. We use the concept of *scheduler* which is a formal device that specifies the order in which processors wake up and messages are delivered. For convenience of analysis, we consider a formal clock: each processor activation step is done at the next moment 1, 2, ... As well, names A and B are given formally to the terminals. However, those clock and names are not available to processors, and are in no sense related to the algorithm.

Notice that an upper bound, confirmed by showing an algorithm, must be valid for *all* schedulers and inputs, while to obtain a lower bound L, it is sufficient to show that, for any algorithm, there *exists* some scheduler and some input, s.t. the execution under that scheduler with that input requires at least L bits sent.

We assume, following [3, 5], that a scheduler wakes each terminal, eventually, if not woken by a message previously, and that no internal processor is awaken by the scheduler. Another important case, when a scheduler may wake any set

of processors, and is ought to wake at least one arbitrary processor, remains as an interesting open question, for both problems addressed in this paper.

A distributed algorithm solving some task is said to have the *terminating property* (or is *terminating*), if each processor becomes eventually ensured that no more message concerning this task will be sent by any processor and that there is no messages in transit, except, may be, messages sent by itself. This property enables each involved processor to begin eventually communicating with other participating processors on another task.[1] For a distributed task T, we denote by $BitC^t(T)$ its bit complexity for the case when the termination property is required; clearly, $BitC^t(T) \geq BitC(T)$.

Suppose that there is an algorithm solving task T, for the case when ids are taken from the set $Z_M = [0..M-1]$. Notice that its slight variation solves it, with the same complexity, if the id range is $[s..s+M-1]$. Indeed, it suffices to simulate the original algorithm, relating to each id x as to $x-s$. In what follows, we identify such a version with the original algorithm.

2 `Leader` and `MaxF` for Two Processors

In this section, we consider the network consisting of two linked processors. We find the bit complexity for `Leader` and `MaxF` and describe all optimal algorithms.

Theorem 1. *For the two processor network,* $BitC(\texttt{Leader}) = BitC(\texttt{MaxF}) = 2\lceil\log_2((M+2)/3.5)\rceil$ *and* $BitC^t(\texttt{Leader}) = BitC^t(\texttt{MaxF}) = 2\lceil\log_2((M+1)/3)\rceil$.

The theorem is implied by the following algorithms and lower bounds. Since `MaxF` is a special variant of `Leader`, we present our upper bounds for `MaxF` only and lower bounds for `Leader` only.

2.1 Algorithms

Proposition 1. *For the two processor network, there exists a terminating algorithm solving* `MaxF` *in* $2\lceil\log_2((M+1)/3)\rceil$ *bits sent.*

Proof. We suggest a recursive algorithm; its execution is divided into rounds sending two bits each. For the minimum value 2 of M, there is a single degenerate round, requiring no communication: each processor decides leader or non-leader according to the value of its id 1 or 0, respectively. For $M \geq 3$, let us divide the id range into the *median* id $\lfloor M/2 \rfloor$, the sub-range of *lesser* ids $[0..(\lfloor M/2 \rfloor - 1)]$, and that of *greater* ids $[(\lfloor M/2 \rfloor + 1)..(M-1)]$. Note that the longest one of the id sub-ranges is of the length $\lfloor M/2 \rfloor$, for any M.

During describing the execution, we identify a processor with its id. For example, we recommend to the reader to consider the cases $Z_M = [0..4]$ ($\lfloor M/2 \rfloor = 2$)

[1] Indeed, the only messages remaining to deliver, if any, are from itself. Hence, by the FIFO property, any *new* message sent by it will reach the target processor after it would finish all its activity in the previous task.

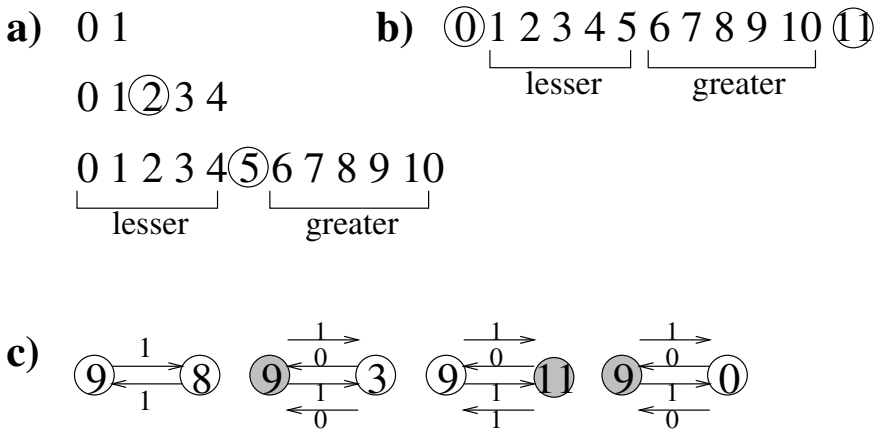


Fig. 1. Optimal algorithms: (a) the terminating case, (b) the non-terminating case, (c) various rounds, for the non-terminating case (earlier messages up; the leader is filled grey)

and $Z_M = [0..10]$ ($\lfloor M/2 \rfloor = 5$). In parentheses, a mnemonic meaning of the operation is given, for easier understanding. For illustration see Fig. 2.1.

Round Description

1. Each lesser id sends 0 [Am I the non-leader?], each greater id sends 1 [Am I the leader?], while the median id sends nothing and waits.
2. If a lesser (resp., greater) id gets 0 (resp., 1), then [it realizes that both ids are lesser (resp., greater) ones, and] the algorithm continues to the next round, for solving the problem in the corresponding sub-range (whose length is at least 2, by the case assumption).
   When the median id gets bit 0 or 1, it decides that it is the leader or non-leader, respectively, and responds by the opposite bit [I agree with your suggestion].
   If a lesser id gets 1, then it decides non-leader; if a greater id gets 0, then it decides leader.

The correctness of the proposed algorithm is immediate. The algorithm is terminating, since at any its stage, any processor knows for sure whether it waits for the next bit to be delivered, or the algorithm finishes.

Let us analyze now the number of bits sent. We define the following recurrent relation:

$$m^t_{2(i+1)} = 2m^t_{2i} + 1,\ i \geq 0;\ m^t_0 = 2\ .$$

Lemma 1. *If M is at most m^t_{2i}, the worst case number of bits sent is $2i$.*

Proof. We prove by induction on i. In the base case $i = 0$, the statement is trivially correct. Assume it is correct for $i = k$, $k \geq 0$, and consider the case

$i = k + 1$. For any $M \leq m^t_{2(k+1)}$, either there is a single round, with 2 bits sent, or the algorithm continues to the same problem in some id sub-range. In the latter case, the length of both id sub-ranges is at most m^t_{2k}, hence, the algorithm finishes the problem solution with sending at most $2k$ bits, by the induction assumption, i.e. $2k+2$ bits are sent totally. The statement correctness follows. □

The solution to the above recurrence relation is:

$$m^t_{2i} = 3 \cdot 2^i - 1 \ .$$

By inverting it, we obtain the worst case number of bits sent be $2\lceil \log_2((M + 1)/3)\rceil$, as required. (Note that, in comparison with the bound $2\lceil \log_2 M\rceil - 2$ of the algorithm [3, 4], this is less by approximately $2 \log_2 1.5 \approx 1$ bit, as the asymptotic average.) □

Proposition 2. *For the two processor network, there exists an algorithm solving* `MaxF` *in* $2\lceil \log_2((M + 2)/3.5)\rceil$ *bits sent.*

Proof. The algorithm is similar, in its structure, to the terminating algorithm as above. Moreover, it coincides with it for M at most 5. Beginning from $M = 6$, the algorithm is as follows. The id range is divided into the *minimum* id 0, the *maximum* id $M - 1$, the *lesser* ids from 1 to $\lfloor M/2\rfloor$, and the *greater* ids from $\lfloor M/2\rfloor + 1$ to $M - 2$. Each round sends 2 bits, except for the finishing round considering the id sub-range of length at least 6, which sends 4 bits.

Round Description, $M \geq 6$

1. Each lesser id sends 0 [Am I the non-leader?], each greater id sends 1 [Am I the leader?], while the minimum and maximum ids decide non-leader and leader, respectively, send nothing and wait.
2. If a lesser (resp., greater) id gets 0 (resp., 1), then [it realizes that both ids are lesser (resp., greater) ones, and] the algorithm continues to the next round, solving the problem in the corresponding sub-range.
   When the minimum or maximum id gets any bit, it responses by the opposite bit [I am not as you are; let us finish].
   If a lesser (resp., greater) id gets 1 (resp., 0), then it sends once more 0 (resp., 1) [confirms its intention].
3. After getting the second bit, the minimum id always responses 0 [I am the non-leader], and the maximum one 1 [I am the leader].
   If a lesser or greater id gets the second bit 0, it decides leader, and if 1 non-leader.

Let us prove that the algorithm solves `MaxF`. The only non-trivial case concerns the decision of a lesser (resp., greater) id which gets the first bit 1 (resp., 0). Let us consider such a lesser id x (the other case is similar). If the other id is a greater or the maximum one, it sends the second bit 1 to x, and x correctly decides non-leader, while if the other id is the minimum one, it sends 0 to x, and x correctly decides leader.

The algorithm is not terminating in the case, when in some round, the minimum or maximum id, w.r.t. the range of that round, decides and waits. Indeed, let that id be x. If the other id is the maximum or minimum one, the execution finishes, and no bit will be ever delivered to x. In all other cases, some bit should be delivered to x, eventually. Note that there are no means, to x, to distinguish between these two situations.

The analysis of number of bits sent has the same structure as that of the terminating algorithm. Therefore, we mention here only the different considerations needed. We define the recurrence relation:

$$m^{nt}_{2i+2} = 2m^{nt}_{2i} + 2,\ i \geq 1;\ m^{nt}_2 = 5, m^{nt}_0 = 2\ .$$

Lemma 2. *If M is at most m^{nt}_{2i}, the worst case number of bits sent is $2i$.*

Proof. The proof is similar to that of Lemma 1, so only difference from that proof is mentioned. The (trivial) base cases of induction are $i = 0$ and $i = 1$. At the induction step, M is at least 6, so the bound to prove is at least 4. The analysis is not similar only for the finishing round. By the round description, the number of bits sent in such a round is exactly 4, which suffices. □

The solution to the above recurrent relation is:

$$m^{nt}_{2i} = 3.5 \cdot 2^i - 2,\ i \geq 1\ .$$

By inverting it, we obtain the worst case number of bits sent be $2\lceil \log_2((M + 2)/3.5)\rceil$, as required. (Note that this is less by approximately $2\log_2 1.75 \approx 1.5$ bits, as the asymptotic average, than the number of bits $2\lceil \log_2 M\rceil - 2$ of the algorithm [3, 4].) □

2.2 Lower Bounds

Now, we pass to the lower bounds. In fact, their proof given below shows that the only structure of an optimal algorithm must be such as that of the algorithm suggested above. We prove the lower bounds for executions under the *symmetric scheduler* [3, 4], defined as follows:

- At step 0, both processors are woken up;
- At each following step, the first bit in every queue, if any, is delivered.

Proposition 3. *For any algorithm solving* `Leader` *in the two processor network and its executions under the symmetric scheduler, at least $2i$ bits must be sent, in the worst case, if ids are chosen from any set of cardinality $M \geq m^t_{2(i-1)} + 1$, when the termination property is required, or $M \geq m^{nt}_{2(i-1)} + 1$, otherwise, $i \geq 1$.*

Proof. Let us consider an an arbitrary integer set Z of cardinality M, as in the statement, and an arbitrary algorithm $\mathcal{A}$ solving `Leader` in the two processor network, when ids are chosen from Z. Since the scheduler is fixed, any choice of ids implies a unique execution. We will call an id *passive*, if it sends no bit, upon its wake-up.

Lemma 3. *1. There exist at most two passive ids.*
2. If there exist two passive ids, then each one of them decides immediately upon its wake-up, while the decision depends on the id only and the decisions for those two passive ids are different.
3. If $\mathcal{A}$ is terminating and M exceeds 2, then there exists at most one passive id.

The proof is omitted. □

Let us, first, assume $\mathcal{A}$ be terminating. We prove the statement of Proposition by induction on i. Basic case $i = 1$: Since M is at most $m_0^t + 1 = 3$, by Lemma 3(3), there are at least two *non-passive* ids. Let us give them to the processors. Since each one of them sends at least one bit, at least two bits are sent.

Assume now correctness of the statement for $i = k$, $k \geq 1$, and let us prove it for the case $i = k + 1$. Among at least $m_{2k}^t + 1$ ids in Z, at most one is passive. Let us divide the set of other ids, which is of cardinality at least m_{2k}^t, into two groups Z_0 and Z_1, according to their first bit sent be 0 or 1, respectively. The largest one of them is of cardinality at least $m_{2(k-1)}^t + 1$, by definition of m_{2k}^t; we denote it by Z'.

Let us consider the continuation of $\mathcal{A}$, from its step 1 an on, for all choices of ids from Z'. In fact, the only information available to each processor after step 0, is its id and the fact that the other id belongs to Z' as well. Therefore, a simulation of $\mathcal{A}$ from step 1 and on is an algorithm solving `Leader`, when ids are chosen from Z'. By the induction assumption, such a solution requires to send at least $2k$ bits, in the worst case. Totally, $\mathcal{A}$ sends at least $2k + 2$ bits, in the worst case, as required.

Let us turn now to the case when the termination property is not required. The proof has the same structure and goes in the same way as the above proof. The only exception is the case when there are two passive ids, M equals $m_{2(i-1)}^{nt} + 1$, and besides either $i = 1$ or $|Z_0| = |Z_1| = m_{2k}^{nt}$. We omit the proof for this case. □

3 `Leader` in a Chain of Known Even Length

Consider `Leader` problem in a chain of length n, where the value of n is known to the processors. Let us denote the middle processor P_0. We call *A-half* and *B-half* the chain intervals $[A..P_0]$ and $[P_0..B]$, respectively.

3.1 Optimal Algorithms

Let us begin from the description of a few algorithms (for illustration see Fig. 3.1). In any execution, we call an internal processor woken by the bit b an *b-processor*.

Algorithm-2, working for $n = 2$: The terminals decide "non-leader", while the single non-terminal has the built-in decision "leader".

Algorithm-4, working for $n = 4$: Terminals decide "non-leader" and send 1; any 1-processor decides "non-leader" and sends 0 forward; the single 0-processor

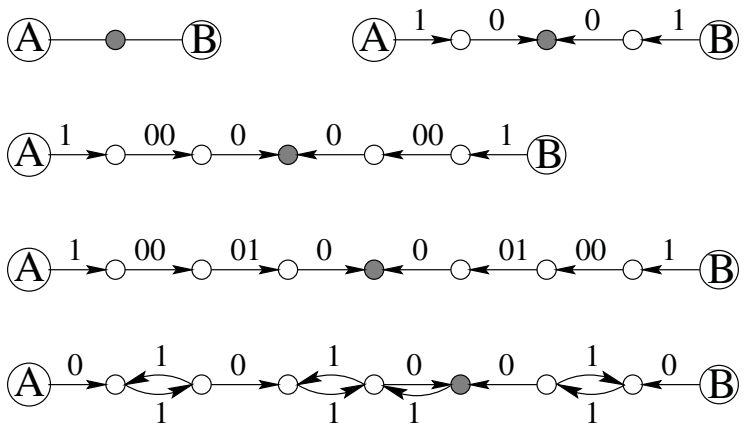


Fig. 2. Optimal algorithms: Algorithm-2; a version of each one of Algorithm-4, Algorithm-6, and Algorithm-8, each with its unique execution; Main Algorithm, with a sample execution. (Bits sent at a link are ordered from left to right. The leader is filled grey.)

P_0 decides "leader" and sends nothing. When it gets one more 0, it does not react.

The flip of bit values 0 and 1 results in another version of Algorithm-4.

Algorithm-6, working for $n = 6$: Each terminal decides "non-leader" and sends 1. Any 1-processor decides "non-leader", and sends 00 forward. Any 0-processor does nothing and waits for an additional bit. Such a processor, after receiving the second bit from the *same* processor, decides "non-leader" and sends 0 forward. If it receives the second bit from the *other* processor, it decides "leader" and sends nothing (this happens for P_0 only).

The rearrangement between links to send 1 and 00 define another versions of Algorithm-6. More algorithm versions arise by changing 00 to 01. Even more versions are the result of the bit values flip, as above.

Algorithm-8, working for $n = 8$: A similar family of algorithms differs from Algorithm-6 in the following: Instead of 1, 00, and 0 sent on the three consequent links of the half-chain, there are 1, 00, 01, and 0 sent on its four links.

The versions of Algorithm-8 are obtained by all the rearrangements of 1, 00, and 01, and by the bit values flip.

Main Algorithm [5], working for $n \geq 8$: Each terminal decides "non-leader" and sends 0. Any 0-processor sends 1 forward. Any 1-processor decides "non-leader", sets that the leader is farther, and sends 0 forward and 1 backward. Upon receiving 1 from its following processor, a 0-processor decides "non-leader", sets that the leader is farther, and sends nothing. Upon receiving 0 from its following processor, a 0-processor decides "leader", and sends nothing.

The second version of Main Algorithm arises by flipping the bit values.

Proposition 4. *Algorithms-2, -4, -6, -8, and Main Algorithm solve Leader in 0, 4, 8, 12, and* $1.5n$ *bits sent, respectively.*

Proof. For Algorithms-2, -4, -6, and -8, the proof is trivial. Let us consider Main Algorithm. Since n is even, the wave meeting happens *always* at a link between a 0-processor, sending 1, and either a 1-processor or a terminal, sending 0. Hence, (i) exactly one 0-processor (that incident to the wave meeting link) gets reply 0, which causes it to decide "leader", and (ii) the number of bits sent at the links looks *always* as two sequences 1,2,...1,2, beginning from the two chain ends, which results in $1.5n$ bits. □

3.2 Lower Bound

Let us consider the family of schedulers which wake A first, allow B to be waken by the first message sent to it, if exists, and otherwise wake B after all processors have become quiescent. Let us call the execution prefix before waking B, under any such scheduler, a *full A-wave*; a *full B-wave* is defined similarly. A full wave is called *halting* if it halts before reaching the other terminal. Clearly, A- and B-waves depend only on the id given to the corresponding terminal. Any prefix of a full wave is called a *wave*. We say that *waves meet* when the first message sent by one of the waves reaches a processor activated in the other wave; we denote the wave meeting moment, if any, by t_m. At that moment, the link, e, carrying the message as above may carry also a message sent by the reached processor; we say that the waves meet at e. It is easy to see that *any* execution begins from *interleaved independent A- and B-waves*, up to their meeting or halting of both of them before a meeting. Suppose that full A- and B- waves, executed separately, cover overlapping chain areas. Clearly, by fine tuning the wave interleaving by the scheduler, we are able to make the waves meet at any link in the overlapped area. We use words like "forward", " backward", "farther", etc. w.r.t. the wave origin.

In this section, we prove the following statement:

Theorem 2. *For* $M \geq 6$, *BitC*(`Leader`) *in an even length chain is* 0 *if* $n = 2$, 4 *if* $n = 4$, 8 *if* $n = 6$, *and* $1.5n$ *if* $n \geq 8$, *even if the value of* n *is known to the processors. Moreover, for* $n \geq 10$, *Main Algorithm is the single optimal algorithm.*

In what follows, we analyze an arbitrary optimal `Leader` algorithm; by Proposition 4, it sends no more bits than stated in Theorem 2, in the worst case.

Consider the case $n = 2$, when no bit is sent. No terminal may decide "leader", since then P_0 would not know the direction to the leader. Hence, Algorithm-2 is the unique optimal algorithm.

Henceforth, we assume that n is at least 4. Let us begin with a few observations. It may be shown that no built-in decision at internal processors is possible, that is *all internal processors should be woken eventually.*

Notice that for at most one id, *ex-id*$_1$, there exists a wave initiated by it, which halts before P_0. Indeed, otherwise, we could give two such ids to A and

B, and arrive at a deadlock without P_0 decided. For the wave from any id other than *ex-id*$_1$, we call its part up to waking P_0 the *half-wave*.

Let us say that a non-terminal processor is a *one-entry*, w.r.t. some execution, if it receives *exactly* one message (waking it) during that execution. Observe that at any execution, where N bits are sent, there are at least $2n-2-N$ one-entries. Indeed, suppose that there are k one-entries, each receiving one bit, while the $n-1-k$ other processors receive at least two bits each. Then, $N \geq k+2(n-1-k) = 2n-2-k$, and the statement follows. As a consequence, for any execution of any optimal algorithm, by Proposition 4, there are at least two one-entries, if $n = 4, 6, 8$, and more than two if $n \geq 10$.

We say that a scheduler is *TPB* (*Tail-Preference Balanced*), if before the wave meeting, it prefers to deliver messages at links closer to the wave origin, and at the links at the same distance from it, to processors closer to it. Let us fix an arbitrary TPB scheduler, $\mathcal{S}$. Observe that, under $\mathcal{S}$ and for ids other than *ex-id*$_1$, the two half-waves should meet at P_0. Moreover, then, all message queues at links are empty at moment t_m, except for those from P_0. Hence, after t_m, information propagates *from* P_0 *only*.

Lemma 4. *Consider an algorithm sending at most N bits. Then, for any id, x, given to w.l.o.g. A, except for ex-id*$_1$ *and at most one more id, ex-id*$_2$*, some other id, y, may be given to B, so that in the execution under $\mathcal{S}$, there are at least $n-1-\lfloor N/2 \rfloor$ one-entries in the A-half-chain.*

The proof is omitted.

Note that for any optimal algorithm, the number of one-entries as in Lemma 4 is at least 1. We assign to each id x as in Lemma, the farthest one-entry in A-half-chain, w.r.t. the execution, E_x, as in Lemma 4; let us denote it by P_x. By the definition of a one-entry, in E_x, the propagation from P_0 after t_m to the direction of x stops before reaching P_x. Hence the interval $[A..P_x]$ has finished all its activity in E_x, including processor decisions, before the A-wave propagated beyond P_x (that is, independently on the information propagated from B). Besides, *any* A-half-wave from x under $\mathcal{S}$ *coincides* with that in E_x. This allows to prove the following:

Corollary 1. *For any id, x, given to w.l.o.g. A, except for ex-id*$_1$*, ex-id*$_2$ *and at most one more id, ex-id*$_3$*, at any execution under $\mathcal{S}$, independently on the id given to the other terminal, all processors in $[A..P_x]$ decide "non-leader" (in particular, the terminal decides so).*

In what follows, we assume that no id is one of *ex-id*$_{1,2,3}$.

Clearly, the reaction of all internal processors to the bit waking it—deciding and bit sending—is completely defined by that bit, since the initial processor state is unique. Let us classify algorithms by properties $\mathcal{I}(b)$, $b = 0, 1$: *A b-processor decides and relays a bit forward immediately upon its wake-up*, of their executions under $\mathcal{S}$.

Case 1: Neither $\mathcal{I}(0)$, nor $\mathcal{I}(1)$ are satisfied.

Such an algorithm cannot be optimal, since then, at the execution under $\mathcal{S}$, each processor would receive at least two messages (either for deciding or for propagating the wave to the direction of P_0), i.e. at least $2n-2$ messages totally.

Case 2: Both $\mathcal{I}(0)$ and $\mathcal{I}(1)$ are satisfied.

We may assume w.l.o.g. that any 1-processor decides "non-leader"; indeed, if both 0- and 1-processors decide "leader", all internal processors would be leaders. Then, each 0-processor decides "leader"; indeed, otherwise, Corollary 1 implies that, under $\mathcal{S}$, no leader would be chosen. Assume further that any 1-processor relays bit 0 farther. If $n = 4$, we arrive at Algorithm-2. Otherwise ($n \geq 6$), each half-wave would contain at least one 0-processor, which decides "leader",—a contradiction.

The only remaining case is that any 1-processor decides "non-leader" and relays 1 farther. If two distinct ids cause the terminal to send 1, there would be no 0-processors, i.e. no internal leaders. Then under $\mathcal{S}$, by Corollary 1, no leader would be elected. If two distinct ids cause the terminal to send 0, there would be two leaders. Hence, in case $M \geq 6$, there will always be an id pair causing a contradiction.

Therefore if $n \geq 6$, the only possible case for an optimal algorithm is:

Case 3 (main): W.l.o.g., $\mathcal{I}(1)$ is satisfied, while $\mathcal{I}(0)$ is not.

Notice that any one-entry is a 1-processor, since an one-entry must decide and relay a message forward immediately. By the above bound $2n-2-N$ for the number of one-entries, in any execution of an optimal algorithm under $\mathcal{S}$, there are at least two one-entries. Hence, each 1-processor must decide "non-leader".

Assume, at-first, that a 1-processor relays 1 farther, upon its wake-up. By Corollary 1, for any id pair x, y and under $\mathcal{S}$, the only region, where the leader may be chosen, is strictly between P_x and P_y. However, both P_x and P_y are 1-processors, which immediately guarantee that all processors in that interval are 1-processors, and hence no leader will be elected,—a contradiction.

Now, we are left with the single possibility (that used in the algorithms mentioned in the Theorem): Any 1-processor relays 0 farther. At-first, assume $n \geq 10$, and consider the execution, E, of an optimal algorithm under $\mathcal{S}$, for any pair of ids, x and y, distinct from $ex\text{-}id_1$. Then, by the bound $2n - 2 - N \geq 3$ for the number of one-entries, w.l.o.g. A-half-chain contains at least two one-entries, while x is given to A, and B sends a message upon its wake-up. Let P and Q denote the first two one-entries in A-half-chain, and k denote the distance between them, $k \geq 2$. Let us change E to another execution prefix, E' which:

- begins with waking A and B,
- does not deliver the first message sent by B until A-wave reaches B or halts,
- coincides with E until delivering the first bit relayed by Q (recall that at this moment the entire interval $[A..Q]$ is quiescent, by definition of the TPB property),
- continues after Q exactly as after P, periodically, with period k links, up to reaching B.

We claim that the obtained execution prefix is legal. This is so, since the situation in each period after Q is exactly the same as at that after P, with no influence from outside.

Let us prove that k equals 2. Assuming $k > 2$, to the contrary, we will show that the execution prefix E' sends too many bits. In $(A..P)$, all processors receive at least two bits each. In each period, all processors except of just one receive so as well. In the finishing part, all links bear at least one bit, except for the last link, bearing at least two messages: to and from B. An easy counting results in more than $1.5n$ bits totally. Hence, k equals 2. More counting shows that exactly three bits should to be sent in every period.

In analyzing the structure of such a period, we show that the two messages to its single non-one-entry come from different directions, and that both contain bit 1. By showing that $[A..P]$ should consist of a single link bearing a single bit 1 from A to P, we arrive finally at Main Algorithm.

For the case $n = 8$, we assume that there exists an algorithm sending *strictly less* than $1.5 \cdot 8 = 12$ bits, in the worst case. Then, the inequality $2n - 2 - N \geq 3$ holds as well, so we are able to analyze it in the same way as in the above case. We thus show that $1.5 \cdot 8$ are sent, arriving at a contradiction.

Now, we consider the case $n = 4, 6, 8$, returning to analysing an *arbitrary* optimal algorithm. We would need another technique. If an algorithm admits *at least one* non-halting wave, that wave may be shown to be periodic, and the algorithm to coincide with Main Algorithm, in the way as above. The following statement finishes the proof of Theorem 2.

Proposition 5. *In the case $n = 4, 6, 8$, if an algorithm generates halting waves only, then it either coincides with one of Algorithms-4, -6 and -8, or sends more bits, than those algorithms, in the worst case.*

Remark: We have very strong reasons to believe that also in the case $n \leq 8$, there is no optimal algorithm other than Algorithm-2, Algorithm-4, Algorithm-6, and Algorithm-8.

References

1. Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* McGraw–Hill, England, 1998.
2. Martin Dietzfelbinger. The linear-array problem in communication complexity resolved. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997, 373–382.
3. Ye. Dinitz, S. Moran, and S. Rajsbaum. Bit complexity of breaking and achieving symmetry in paths and rings. In: *Proc. of the 31th Symposium on Theory of Computing, STOC'99*, 265–274.
4. Ye. Dinitz, S. Moran, and S. Rajsbaum. Bit complexity of breaking and achieving symmetry in chains and rings. Technical Report #CS-2004-11, Dept. of Comp. Sci., Technion, August 2004, 29p.
5. Ye. Dinitz, S. Moran, and S. Rajsbaum. Exact Communication Costs for Consensus and Leader in a Tree. *J. of Discrete Algorithms* **1** (2003), 167–183.
6. Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, 1997.
7. Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., 1996.

Finding Short Right-Hand-on-the-Wall Walks in Graphs

Stefan Dobrev[1], Jesper Jansson[2],
Kunihiko Sadakane[4], and Wing-Kin Sung[3]

[1] SITE, University of Ottawa, Canada
sdobrev@site.uottawa.ca
[2] Department of Computer Science and Information Systems,
The University of Hong Kong, Hong Kong
jjansson@cs.hku.hk
[3] School of Computing, National University of Singapore
ksung@comp.nus.edu.sg
[4] Department of Computer Science and Communication Engineering,
Kyushu University, Japan
sada@csce.kyushu-u.ac.jp

Abstract. We consider the problem of *perpetual traversal* by a single agent in an anonymous undirected graph G. Our requirements are: (1) deterministic algorithm, (2) each node is visited within $O(n)$ moves, (3) the agent uses no memory, it can use only the label of the link via which it arrived to the current node, (4) no marking of the underlying graph is allowed and (5) no additional information is stored in the graph (e.g. routing tables, spanning tree) except the ability to distinguish between the incident edges (called *Local Orientation*).

This problem is unsolvable, as has been proven in [9, 28] even for much less restrictive setting. Our approach is to somewhat relax the requirement (5). We fix the following traversal algorithm: *"Start by taking the edge with the smallest label. Afterwards, whenever you come to a node, continue by taking the successor edge (in the local orientation) to the edge via which you arrived"* and ask whether it is for every undirected graph possible to assign the local orientations in such a way that the resulting perpetual traversal visits every node in $O(n)$ moves.

We give a positive answer to this question, by showing how to construct such local orientations. This leads to an extremely simple, memoryless, yet efficient traversal algorithm.

1 Introduction

The problem of searching and exploring an unknown environment is a fundamental problem with applications ranging from robot navigation to searching the WWW. As such, it has been extensively studied under many different assumptions about the environment. Typically, either a geometric setting has been assumed (see e.g. [7, 11, 29]) or the environment is modeled as a graph with moves permitted only along the edges.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 127–139, 2005.

The graph setting has been extensively investigated [1, 3, 6, 12, 13, 14, 16, 20, 25] under many different assumptions (directed vs undirected graphs, anonymous nodes vs nodes with distinct identities) and goals (different variants of the exploration, focusing on time complexity, minimizing the memory requirements).

An important aspect of any solution is its memory requirements – both in the exploring agent(s) and in the network environment itself (can the agent mark the nodes, what is the nature of the marks and how many can be used?). Since the desire is to have simple and cost efficient agents, and there can be many of them independently operating in the network, it is of practical importance to limit both the local memory of the agents, and their ability to mark the network.

An extreme case of minimizing memory requirements is to limit the agents memory to a constant number of bits. This can be modeled as exploring a graph using finite automata and has been intensively studied in 70's [9, 24, 26, 28]. The strongest result is due to Rollik [28]: No finite group of finite automata can cooperatively explore all cubic planar graphs (see [21] for more recent results). This means that we either have to allow the agents to use more memory, resort to randomization or provide some structural information that restricts the set of graphs we have to traverse.

If we do not place strict restrictions on the local memory, single pebble is sufficient to explore the graph [5], even for anonymous directed graphs [1].

Another possibility is to drop the determinism requirement – it is known that a random walk of length $O(n^3)$ would, with high probability, visit every node [22]. Trying to regain determinism led into research of derandomized random walks, searching a sequence (called Universal Traversal Sequence) of edge labels that would traverse all graphs in a given class. While many interesting results have been achieved [2, 4, 23, 27], the memory requirements are not always clear and the traversal times are rather high, especially with respect to our goal of $O(n)$ time.

Research most closely related to our result considers using structural information to improve the time and/or memory complexity of graph traversal. First results (concerning exploration of a labyrinth using a compass) are due to Blum and Kozen [8]. Later, Flocchini et al [18] introduced a more general notion of Sense of Direction and proved that traversal can be performed using $O(n)$ messages/agent moves [17]. Fraigniaud et al [19] have shown that interval routing scheme can be used to achieve the same. In fact, given a spanning tree, the graph can be traversed using $O(n)$ moves. Pelc and Panaite [25] studied the impact of having a map of the graph on the efficiency of graph exploration/traversal. All these solutions, though, use quite a lot of memory – either in the network (routing tables in [19], remembering the spanning tree) or in the agent (storing Sense of Direction and remembering visited nodes in [17], remembering the network map in [25]).

[1] The graph must be strongly connected and an upper bound on the number of nodes must be known. The time complexity, while polynomial, is quite high, though.

In this paper we propose to use the capabilities already present in the system – namely the ability to distinguish the links incident to a node – to store the information allowing efficient traversal: A common requirement in point-to-point networks is that the nodes can distinguish between incident edges (often called *Local Orientation*). This is normally done by giving the edges incident to a node v numbers $1, 2, \ldots, d_v$, where d_v is the degree v. Usually, there is no assumption on how is this edge ordering chosen. In this paper we propose to choose the local orientations in a very specific way. Whether this costs us any memory depends on how are the lower level communication layers implemented, but it is quite conceivable that if done at the time of network construction/initialization, it comes essentially for free.

We fix the following traversal algorithm: *"Start by taking the edge labelled 1. Afterwards, whenever you come to a node, continue by taking the successor edge (in the local orientation) to the edge via which you arrived"* and ask whether it is for every undirected graph possible to to choose the local orientations in such a way that the resulting perpetual traversal visits every node in $O(n)$ moves.

We give a positive answer to this question, by showing how to construct such local orientations. This leads to an extremely simple, memoryless, yet efficient traversal algorithm.

The paper is organized as follows: In Section 2 we introduce the notation used and give basic definitions and properties. Section 3 contains the main result of the paper. In Section 4 we discuss how to adapt to dynamically changing networks. Section 5 contains open questions, as well as a brief comparison to Sense of Direction.

2 Preliminaries

Let G be a simple, connected, undirected graph. Let d_v denote the degree of vertex v in G. We assume that each vertex can distinguish edges incident to it, by having assigned unique label to each incident edge[2]. This labeling (denoted π_v and called *local orientation*) and defines cyclical ordering of the edges incident to v. Let $succ_v(e)$ denote the successor of e in π_v. Denote by $\boldsymbol{G}$ the symmetric directed graph obtained from G by replacing each undirected edge by two directed edges in opposite direction. For each directed edge $\boldsymbol{e} = \{u, v\}$ we define the *underlying* edge to be the undirected edge (u, v).

We want to find a (short, possibly non-simple) cycle $\boldsymbol{C}$ in $\boldsymbol{G}$ containing all vertices of $\boldsymbol{G}$ and satisfying the right-hand rule: If $e_1 = \{u, v\}$ and $e_2 = \{v, w\}$ are two successive edges of $\boldsymbol{C}$ then $e_2 = succ_v(e_1)$. Since the local orientations can be rotated so that the underlying edge with label 1 is used in outgoing direction at every vertex, the algorithm *"Start by using edge labelled* 1 *and then follow the successor edges"* traverses exactly $\boldsymbol{C}$. We call such a cycle a *witness cycle* for G.

[2] More precisely, to endpoints of the edges incident to v. Each edge gets two labels, one at each endpoint. Note that these two labels can be different and unrelated.

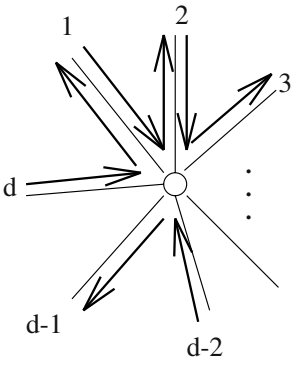


Fig. 1. Ordering two bidirectional, two incoming and two outgoing underlying edges

Let $\boldsymbol{H}$ be a subgraph of $\boldsymbol{G}$ containing all its vertices. For a vertex v, denote by b_v, i_v and o_v the number underlying edges incident to v used by $\boldsymbol{H}$ in both directions, only incoming and only outgoing, respectively. Let d'_v denote the number of underlying edges used by $\boldsymbol{H}$, i.e. $d'_v = b_v + i_v + o_v$.

The overall idea is to find a graph $\boldsymbol{H}$ containing all the edges of a witness cycle $\boldsymbol{C}$ and then to figure out how to label the edges of $\boldsymbol{H}$ to obtain single witness cycle. The following definition captures the right-hand traversal property that must be satisfied at each vertex of a witness cycle:

Definition 1. *We say that a vertex v is* RH-traversable *if there exists a local orientation π_v in v such that for each directed edge of $\boldsymbol{H}$ incoming to v via an underlying edge e there exists an outgoing edge in $\boldsymbol{H}$ leaving v via the underlying edge that succeeds e in π_v.*

We call such ordering a witness *ordering for v.*

If $b_v = d_v$, the vertex v is said to be *saturated*. The following Lemma characterizes the necessary local conditions for existence of the witness ordering:

Lemma 1. *v is RH-traversable if and only if v is saturated or $i_v = o_v > 0$.*

Proof. The if *direction:* If v is saturated, any ordering of the underlying edges will do. In the second case, choose any ordering in which bidirectional edges are labelled $1, 2, \ldots, d_v$, forming one compact block followed by an outgoing edge. All remaining unidirectional edges are placed as a block preceding the bidirectional block; the edges of this unidirectional block alternate directions, with the last edge preceding the bidirectional block being incoming – see Figure 1.

The only if *direction:* First of all, note that since the successor function is injective, i_v must be equal to e_v. Furthermore, note that if a bidirectional underlying edge is followed by an unused underlying edge, the RH-traversal property is violated. Finally, if v is not saturated and there are no unidirectional underlying edges, there must be such bidirectional edge. □

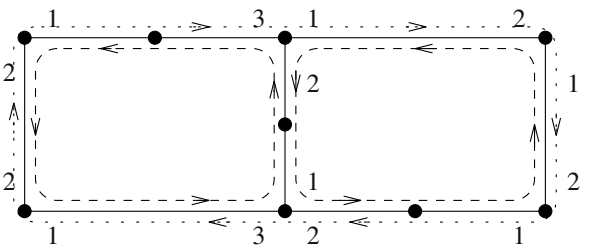


Fig. 2. Each vertex is RH-traversable, but the witness orderings of the vertices define several cycles and no cycle spans the whole graph

Note that if a witness cycle exists, each vertex is RH-traversable. The converse is not necessarily true, as the witness orderings of the vertices might define several cycles, none of which span the whole graph – see Figure 2.

3 Main Result

First note that if G is Hamiltonian, a Hamiltonian cycle can be chosen as witness cycle and we get $\boldsymbol{C}$ of length n. (The RH-traversability is trivially satisfied as each node is visited only once.) Therefore, for the rest of the paper, we assume G is not Hamiltonian.

The main idea of our approach is to

- first find a subgraph $\boldsymbol{H}$ such that each vertex is RH-traversable and
- then figure out how to connect the edges of $\boldsymbol{H}$ to form a single witness cycle $\boldsymbol{C}$.

One obvious possibility is to set $\boldsymbol{H} = \boldsymbol{G}$, i.e. use all edges bidirectionally. From Lemma 1 we know that each vertex can be made RH-traversable. Moreover, since each node of $\boldsymbol{G}$ is of even degree, $\boldsymbol{G}$ has an Eulerian cycle. However, it is not immediately clear how to choose the local edge orderings to satisfy RH-traversability and simultaneously result in a single cycle. Another problem is that the resulting cycle would be of length $O(|E|)$, not $O(n)$.

This forces us to take the following more refined approach:

1. Construct a directed graph $\boldsymbol{H}$ such that
   (a) The undirected graph H' induced by the bidirectional edges of $\boldsymbol{H}$ is connected and contains all vertices of $\boldsymbol{G}$.
   (b) Each vertex v of $\boldsymbol{H}$ is either saturated or has exactly two unidirectional underlying edges, one incoming and one outgoing.
   (c) $\boldsymbol{H}$ contains $O(n)$ edges.
2. For each vertex of $\boldsymbol{H}$ define a witness ordering. (These orderings define one or more cycles in $\boldsymbol{H}$.)
3. Locally modify the orderings in some nodes in order to merge these cycles into one supercycle $\boldsymbol{C}$ containing all vertices, while maintaining RH-traversability.

The first property of $\boldsymbol{H}$ ensures that if we connect all vertices of $\boldsymbol{H}$ into a single cycle $\boldsymbol{C}$, it will span the whole graph. The second property ensures that each vertex is RH-traversable and the third guarantees that $\boldsymbol{C}$ is of size $O(n)$. Note again that the second property does not guarantee by itself that the witness orderings of the vertices define single cycle spanning $\boldsymbol{H}$ (see Figure 2) – we really need to do the third step.

3.1 Constructing Subgraph $\boldsymbol{H}$

The following algorithm constructs the graph $\boldsymbol{H}$:

Algorithm CONSTRUCT $\boldsymbol{H}$:

1: $\boldsymbol{H} \leftarrow \emptyset$; {*Unless stated otherwise, when an edge is added to* $\boldsymbol{H}$*, it is added bidirectionally*}.
2: **repeat**
3: Find in G a cycle C_i such that it does not contain two consecutive vertices that are both in $\boldsymbol{H}$.
4: Add C_i to $\boldsymbol{H}$.
5: **until** no such cycle C_i can be found.
6: Add to $\boldsymbol{H}$ all vertices of G not yet in $\boldsymbol{H}$, together with all their incident edges.
7: If $\boldsymbol{H}$ is not connected, add some bridging edges to make it connected. (Note that because of the previous step, there is no need to add vertices.)
8: Let $\tilde{G} = (\tilde{V}, \tilde{E})$ be the graph induced by all yet unused underlying edges.
9: **repeat**
10: **repeat**
11: Find a vertex v of degree 1 in $\tilde{G}$.
12: Add to $\boldsymbol{H}$ the edge incident to v in $\tilde{G}$; remove v from $\tilde{G}$.
13: **until** There is no vertex of degree 1 in $\tilde{G}$
14: Find in $\tilde{G}$ a cycle C'_i.
15: Add C'_i to $\boldsymbol{H}$ *unidirectionally* (in arbitrary direction) and remove all the vertices of C'_i from $\tilde{G}$.
16: **until** no such cycle C'_i can be found.
17: Add the remaining edges of $\tilde{G}$ (a forest, possibly empty) to $\boldsymbol{H}$.

Lemma 2. $\boldsymbol{H}$ *uses at most* $5n$ *underlying edges.*

Proof. We prove the lemma by observing the following facts:

- **Fact 1.** *The number of edges added on lines 2... 5 is at most* $2n-3k_1$*, where* k_1 *is the number of resulting connected components.*

  Proof: We charge the cost of each cycle to its new vertices. Since at least half of the vertices of the cycle are new, each one of them is charged at most 2 edges. In addition, the vertices of the first cycle (of size at least 3) in each component are charged 1 edge each (as they are all new).
- **Fact 2.** *The number of edges added on line 6 is at most* $2n-2$.

Proof: Let us divide those edges into E_1 – the edges between nodes added in line 6 and E_2 – the edges between the new and the "old" (added before line 6) nodes. The graph induced by edges in E_1 does not contain cycle, otherwise a cycle of all-new vertices would have been found in line 3. Similarly, the graph induced by the edges in E_2 does not contain cycle, otherwise that cycle (containing exactly half new vertices) would have been found in line 3.

– **Fact 3.** *The number of edges added on line 7 is at most $k_2 < k_1$, where k_2 is the number of connected components after line 6.*

Proof: Straightforward.

– **Fact 4.** *The number of edges added on lines 9...17 is at most n.*

Proof: For each edge that is added to $\boldsymbol{H}$ one vertex is removed from $\tilde{G}$. □

Let us call the cycles added in line 15 *relief* cycles.

Lemma 3. *$\boldsymbol{H}$ is connected, contains all vertices of G and each vertex is either saturated or it lies on exactly one relief cycle.*

Proof. The first two properties follow by construction from lines 6 and 7. If node v does not lie on a relief cycle, then either it has never been in $\tilde{G}$ or it was removed from there in lines 10...13 or 17. Either case can happen only if v is (or becomes) saturated. v cannot be in more then once relief cycle, because it is removed from $\tilde{G}$ when its relief cycle is added to $\boldsymbol{H}$. □

Lemma 4. *The complexity of Algorithm* Construct $\boldsymbol{H}$ *is $O(n^3)$.*

Proof. We will show how to implement line 3 (finding a cycle that does not contain consecutive old vertices) in time $O(n^2)$. This straightforwardly results in $O(n^3)$ time for the loop on lines 2..5.

The loop on lines 9..16 can be executed only $O(n)$ times, and the statements in its body can easily be implemented in $O(n^2)$ time. As the remaining steps can be easily implemented in $O(n^2)$ time, the overall complexity would be $O(n^3)$.

The line 3 can be implemented in $O(n^2)$ in the following way: Define graph $G' = (V', E')$ as follows: (1) V' contains all old vertices (the vertices $in\boldsymbol{H}$) and one vertex for each connected component of the graph induced by the new vertices. (2) An edge $(u, v) \in E'$ where u is an old vertex and v corresponds to a connected component of new vertices if and only if there is an edge in G connecting u to a vertex from the connected component corresponding to v.

Note that a cycle in G' defines a cycle in G satisfying the requirements of line 3. Since $O(n^2)$ time is sufficient for constructing G' as well as for finding a cycle in it, line 3 can be implemented in time $O(n^2)$. □

3.2 Constructing Witness Cycle C

Once we have $\boldsymbol{H}$, the local ordering of underlying edges in each vertex is initialized according to the construction from the proof of Lemma 1. We know (from Lemmas 3 and 1) that such witness ordering exists for each vertex v; however, we

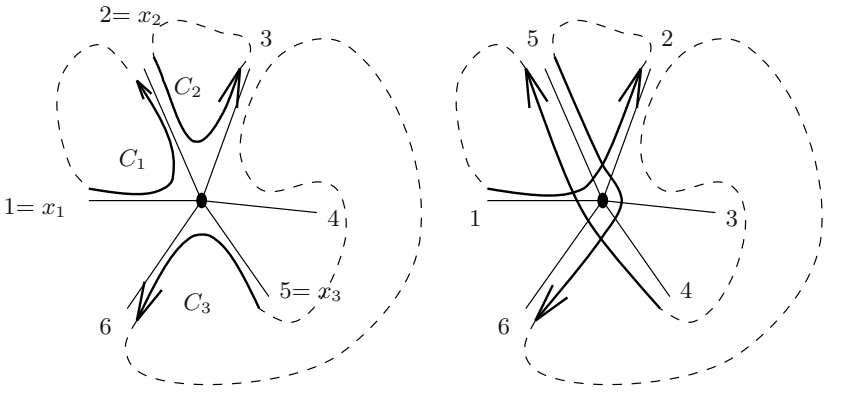


Fig. 3. Applying rule *Merge3*

may not get a single cycle spanning all vertices (see Figure 2). In the next step, we combine the resulting cycles until we get one such cycle, while maintaining RH-traversability. To achieve that, we use the following rules:

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 underlying edges in v containing incoming edges for cycles C_1, C_2 and C_3, respectively (x_1, x_2 and x_3 can be unidirectional or bidirectional). The ordering of the edges in v which makes the successor of x_2 become the successor of x_1, successor of x_3 become the successor of x_2 and successor of x_1 become successor of x_3 and keeps the relative order of the remaining edges the same (see Figure 3) connects the cycles C_1, C_2 and C_3 into one cycle, while remaining a witness ordering for v (because the original ordering was).

Rule EatSmall: Fix an arbitrary ordering γ of the cycles. Let C_1 be the smallest non-simple cycle in this ordering and let v be a vertex appearing in C_1 at least twice which is also incident to a different cycle C_2 such that $\gamma(C_1) < \gamma(C_2)$. Let x and y be underlying edges containing incoming edges of C_1 and C_2 in v, 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 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 remaining edges does not change – see Figure 4.

Lemma 5. *Applying the rule* EatSmall *results in transfer of one loop of edges from cycle C_1 to C_2, while maintaining RH-traversability.*

Proof. Straightforward from construction. □

The overall strategy of applying these rules is as follows:

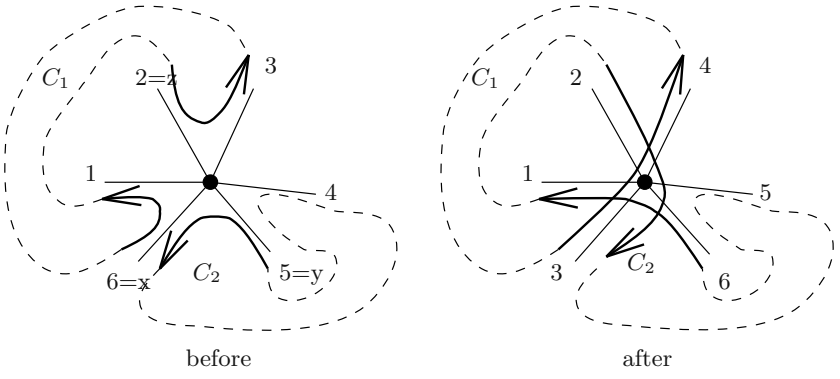


Fig. 4. Applying rule *EatSmall*

Algorithm MergeCycles:

1: **repeat**
2: **while** rule *Merge3* can be applied **do**
3: Apply rule *Merge3*.
4: **end while**
5: Apply rule *EatSmall*.
6: **until** neither *Merge3* nor *EatSmall* can be applied
7: (Optional) remove all simple cycles

Lemma 6. *The algorithm* MergeCycles *terminates in* $O(n^3)$ *time.*

Proof. Note that initially there are at most $10n/3$ cycles ($\boldsymbol{H}$ has at most $10n$ edges and each cycles has at lest 3 edges). Since rule *Merge3* decreases the number of cycles by 2 and rule *EatSmall* does not increase the number of cycles, rule *Merge3* can be applied at most $5n/3$ times during the whole execution of algorithm MergeCycles.

Since rule *EatSmall* transfers come edges from the smallest non-simple cycle to a bigger (in γ) cycle, it can be successively (without intervening *Merge3*) applied only $O(n)$ times (remember, the number of edges is at most $10n$). This means that the rule *EatSmall* can be applied only $O(n^2)$ times.

In order to apply rule *Merge3*, we need to find a vertex incident to three different cycles. In order to apply rule *EatSmall*, we need to find the smallest non-simple cycle and a repeated vertex on this cycle which is incident to a bigger cycle. Both tests can be straightforwardly done in $O(n)$ time by traversing and marking the different cycles, resulting in $O(n^3)$ overall complexity for algorithm MergeCycles. □

Lemma 7. *If neither* Merge3 *nor* EatSmall *can be applied,* $\boldsymbol{H}$ *consists of a single non-simple cycle spanning all the vertices and of a set of pairwise vertex disjoint simple cycles.*

Proof. Before proceeding with the proof, let us remind you that the graph H' consisting of bidirectional edges of $\boldsymbol{H}$ is connected and contains all vertices of $\boldsymbol{H}$. (Follows directly from lines 6 and 7 of the construction of $\boldsymbol{H}$.)

Let C_1 be the smallest non-simple cycle at the moment when neither rule can be applied. Let E' be the set of all underlying edges which are not used by C_1, but are incident to C_1. Each edge of E' is used by a single cycle, otherwise rule *Merge3* could be applied.

Assume first that all these edges are unidirectional. Then all edges of H' are used by C_1, because H' is connected and E' would separate it. Since H' contains all vertices, C_1 does as well. No underlying bidirectional edges outside C_1 means that all other cycles are pairwise edge-disjoint. However, they must be also vertex disjoint, because the rule *Merge3* cannot be applied and C_1 contains all vertices. Similarly, all other cycles are simple, since C_1 contains all vertices and rule *EatSmall* cannot be applied.

To complete the proof, we prove by contradiction that there is no bidirectional edge incident to C_1, but not belonging to C_1. Assume the opposite. From the properties of H' we get that either there is a vertex $v \in C_1$ incident to both an external bidirectional edge and a bidirectional edge in C_1 (contradiction, as that would allow rule *EatSmall* to apply, since the outside bidirectional edge can only belong to a larger non-simple cycle) or that each of the vertices of C_1 is incident to an outside bidirectional edge (in such case either C_1 is simple cycle or *EatSmall* can be applied – contradiction in both cases). □

Now we are ready for the main theorem:

Theorem 1. *There exists a witness cycle* $\boldsymbol{C}$ *of length at most* $10n$ *covering all vertices of* G.

Proof. From Lemma 6 we know that eventually no rule can be applied. From Lemma 7 we get that at that moment there exists single non-simple cycle (which we choose as $\boldsymbol{C}$) covering all vertices. Since this cycle uses each directed edge of $\boldsymbol{H}$ at most once, from Lemma 2 we get the length property. □

Note 1: We can remove from $\boldsymbol{H}$ all the remaining simple cycles to get a graph containing only $\boldsymbol{C}$. The RH-traversability will obviously not be violated.

Note 2: In each vertex we can rotate the edges in such way that the edge labelled 1 will always be in $\boldsymbol{C}$.

From Lemmas 4 and 6 we get the main complexity theorem:

Theorem 2. *Witness cycle of length at most* $10n$ *covering all vertices of* G *can be constructed in time* $O(n^3)$.

4 Adapting to Dynamic Topology Changes

In previous section we described how to initialize the network so that the RH-traversal leads to an efficient traversal. In this section we show how to maintain

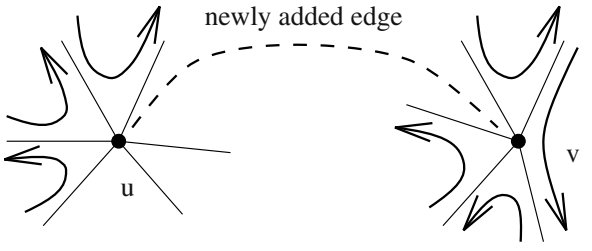


Fig. 5. Adding an edge connecting two unsaturated vertices

this property even in the case of topology changes. More precisely, we show how to modify the local orientations in case of adding new vertices and edges to the network.

In order to simplify the algorithm, we assume the only topology changes are (1) connecting a new vertex to the existing graph by a single edge, and (2) adding an edge between two existing vertices. More complex changes can easily be implemented using a sequence of these basic operations.

If a new edge (u, v) connects two unsaturated vertices, it can be inserted between the outgoing and incoming underlying edges without violating RH-traversability – see Figure 5. However, if one of the vertices is saturated, we have to use the new edge in both directions. Inserting the edge at position 1 ensures that it is a successor for some incoming edge, and that it has a successor outgoing edge, i.e. both u and v remain RH-traversable. However, this might result in splitting $\boldsymbol{C}$ into two cycles. That can be easily corrected by applying rule *EatSmall* while possible, as we known that *Merge3* is not applicable. Note that it is sufficient to perform the test only at node u, as we know that if there are indeed two cycles, they meet at u and v.

Algorithm ADAPT:

1: { *Edge* (u, v) *(and possibly a new vertex* v*) has been added.*}
2: **if** either u or v are saturated **then**
3: Insert (u, v) as a an edge used bidirectionally in $\boldsymbol{C}$ to location 1 in the local orientations of u and v.
4: **else**
5: Insert (u, v) as an edge unused in $\boldsymbol{C}$ to a place between outgoing and incoming underlying edges in u and v – see Figure 5.
6: **end if**
7: Apply rule *EatSmall* at u while possible.
8: (Optional) Remove all non-Hamiltonian simple cycles from.

By construction and from Lemma 7 we get:

Theorem 3. *Applying algorithm* ADAPT *after each topology change will maintain* $\boldsymbol{C}$ *as the witness cycle containing every node of the graph. Moreover, at most* 2 *directed edges are added to* $\boldsymbol{C}$ *for each edge newly added to* G.

After adding n' new vertices and m' new edges, the resulting witness cycle is guaranteed to grow by no more then $2m'$ edges. If m' becomes too high, recomputing the witness cycle might be necessary to bring the length back to $O(|V|)$. Our approach does not handle vertex and/or edge removal, as the graph could become disconnected and/or severe non-local changes might be needed.

5 Conclusions

We have shown that for every connected simple undirected graph the local orientations in the vertices can be chosen in a way that creates a right-hand rule cyclical walk of length at most $10n$ covering all vertices. Moreover, we have shown how to maintain this property even when more vertices and edges are added to the graph. Still, several questions remain unanswered:

- Can the length of the walk be further reduced? What is the lower bound?
- Can the time complexity of finding a witness cycle of length $O(n)$ be reduced from $O(n^3)$? How?
- What is the time complexity of finding the *shortest* witness cycle? How to find it?
- The only property of the walk we were interested in was its length. Suppose we want to use these walks for mutual search [10] instead of traversal. How do we design the local orientations so that performing RH-walk leads to efficient mutual search?
- How to compute the local orientations in a distributed manner? What can be done if the nodes are anonymous?
- How to react to node or edge removal?

We can view our construction as a way to create globally consistent edge labelling. Comparing it to another globally consistent edge labelling, namely Sense of Direction, we observe that our approach uses minimal number of different labels and allows much simpler and more memory efficient graph traversal. However, Sense of Direction is more general and can be used in ways our construction cannot (e.g. avoiding entering a node - see [15].)

References

1. S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing,* 29:1164–1188, 2000.
2. N. Alon, Y. Azar, and Y. Ravid. Universal sequences for complete graphs. *Discrete Appl. Math.*, 27(1-2):25–28, 1990.
3. B. Awerbuch, M. Betke, and M. Singh. Piecemeal graph learning by a mobile robot. *Information and Computation*, 152:155–172, 1999.
4. A. Bar-Noy, A. Borodin, M. Karchmer, N. Linial, and M. Werman. Bounds on universal sequences. *SIAM J. Comput.*, 18:268–277, 1989.
5. M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proc. of STOC 98*, pages 269–287, 1998.

6. M. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. of FOCS 94*, pages 75–85, 1994.
7. A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26:110–137, 1997.
8. M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. of FOCS 78*, pages 132–142, 1978.
9. L. Budach. Automata and labyrinths. *Math. Nachrichten*, pages 195282, 1978.
10. Harry Buhrman, Matthew Franklin, Juan A. Garay, Jaap-Henk Hoepman, John Tromp, and Paul Vitányi. Mutual search. *J. ACM*, 46(4):517–536, 1999.
11. X. Deng, T. Kameda, and C. H. Papadimitriou. How to learn an unknown environment i: The rectilinear case. *Journal of the ACM*, 45:215–245, 1998.
12. X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.
13. A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *Proc. 10th European Symposium on Algorithms (ESA'02)*, pages 374–386, 2002.
14. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51:38–63, 2004.
15. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Finding a black hole in an arbitrary network: optimal mobile agents protocols. In *Proc. of PODC 2002*, pages 153–162, 2002.
16. C.A Duncan, S.G. Kobourov, and V.S.A Kumar. Optimal constrained graph exploration. In *12th ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 807–814, 2001.
17. P. Flocchini, B. Mans, and N. Santoro. On the impact of sense of direction on communication complexity. *Information Processing Letters*, 63(1):23–31, 1997.
18. P. Flocchini, B. Mans, and N. Santoro. Sense of direction: definition, properties and classes. *Networks*, 32(3):165–180, 1998.
19. P. Fraigniaud, C. Gavoille, and B. Mans. Interval routing schemes allow broadcasting with linear message-complexity. *Journal of Distributed Computing*, 14(4):217–229, 2001.
20. P. Fraigniaud and D. Ilcinkas. Digraph exploration with little memory. In *21st Symp. on Theoretical Aspects of Computer Science (STACS'04)*, 2004.
21. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. In *Proc. of MFCS 2004*, pages 451–462, 2004.
22. U. Friege. A tight upper bound on the cover time for random walks on graphs. *Random Structures and Algorithms*, 6(1):51–54, 1995.
23. S. Hoory and A. Wigderson. Universal traversal sequences for expander graphs. *Inf. Process. Lett.*, 46(2):67–69, 1993.
24. D. Kozen. Automata and planar graphs. In *Proc. of Fundations Computatial Theory (FCT 79)*, pages 243–254, 1979.
25. P. Panaite and A. Pelc. Impact of topographic information on graph exploration efficiency. *Networks*, 36.
26. M.O. Rabin. Maze threading automata. Technical Report Seminar Talk, University of California at Berkeley, October 1967.
27. O. Reingold. Undirected st-connectivity in log-space. *Electronic Colloquium on Computational Complexity*, 94, 2004.
28. H.A. Rollik. Automaten in planaren graphen. *Acta Informatica*, 13:287–298, 1980.
29. N. Roo, S. Hareti, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of length,non-heuristic algorithms. Technical Report ORNL/TM12410, Oak Ridge National Lab., 1993.

Space Lower Bounds for Graph Exploration via Reduced Automata*

Pierre Fraigniaud[1], David Ilcinkas[1], Sergio Rajsbaum[2], and Sébastien Tixeuil[1]

[1] CNRS, LRI, Univ. Paris Sud, France
[2] Instituto de Matemáticas, UNAM, D.F. 04510, Mexico
{pierre, ilcinkas, tixeuil}@lri.fr
rajsbaum@math.unam.mx

Abstract. We consider the task of exploring graphs with anonymous nodes by a team of non-cooperative robots modeled as finite automata. These robots have no *a priori* knowledge of the topology of the graph, or of its size. Each edge has to be traversed by at least one robot. We first show that, for any set of q non-cooperative K-state robots, there exists a graph of size $O(qK)$ that no robot of this set can explore. This improves the $O(K^{O(q)})$ bound by Rollik (1980). Our main result is an application of this improvement. It concerns exploration with stop, in which one robot has to explore and stop after completing exploration. For this task, the robot is provided with a pebble, that it can use to mark nodes. We prove that exploration with stop requires $\Omega(\log n)$ bits for the family of graphs with at most n nodes. On the other hand, we prove that there exists an exploration with stop algorithm using a robot with $O(D \log \Delta)$ bits of memory to explore all graphs of diameter at most D and degree at most Δ.

Keywords: Graph exploration, finite automaton, robot, mobile agent.

1 Introduction

The problem of exploring an unknown environment occurs in a variety of situations, like robot navigation, network maintenance, resource discovery, WWW search, etc. The environment is modeled as a graph where one or more mobile agents, called *robots* in this paper, are trying to collectively traverse every one of its edges. There is a large body of work, that considers several variants of the problem, since at least 1951; see *e.g.* [1, 2, 3, 4, 5, 6, 7, 8] and references herein.

In this paper we are interested in exploration of undirected graphs where nodes are not uniquely labeled. Besides the theoretical interest of understanding when or at what cost such graphs can be explored, this situation can occur

* This work has been supported by the projects: INRIA "Grand Large", "PairAPair" of the ACI "Masses de Données", "FRAGILE" of the ACI "Sécurité et Informatique," LAFMI (Franco-Mexican lab in Computer Science), and PAPIIT-UNAM.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 140–154, 2005.

in practice, due to *e.g.* privacy concerns, limited capabilities of the robots, or simply anonymous edge intersections. We do assume that a robot can identify the edges incident to a node through unique port labels. Our main goal is to compute complexity bounds on the amount of memory needed by a set of robots as a function of the size of the graphs that they can explore.

A robot moves from one node to another along the edges. When in a node, it (deterministically) decides on the port number of an incident edge to move to the node at the other end of the corresponding edge. It is easy to see that a robot can traverse all edges of some graphs, say a cycle, but that it cannot recognize when it has visited a node twice, so it explores all the graph but never stops. Thus we consider also robots that can mark nodes; as in previous work *e.g.* [1, 2] the robot can drop a pebble in a node and later identify it and pick it up. In this case the robot can explore a graph and stop.

1.1 Collective Exploration

A graph that a set of robots cannot explore when they all star from some given node (or set of nodes) is said to be a *trap* for them. The first trap for a finite state robot is generally attributed to Budach [3] (the trap is actually a planar graph). The trap constructed by Budach is of large size. A much smaller trap was described in [6]: for any K-state robot, there exists a trap of at most $K+1$ nodes. In [7], Rollik proved that no finite set of finite cooperative robots, *i.e.*, automata that exchange information only when they meet at a node, can explore all graphs. In the proof of this result, the author uses as a tool a trap for a set of q non-cooperative K-state robots. This latter trap is of size $O(K^{O(q)})$ nodes. The size[1] $\tilde{O}(K^{K^{\cdot^{\cdot^{K}}}})$, with $2q+1$ levels of exponential, of the trap constructed for cooperative robots depends highly on the size of a trap for non-cooperative robots.

In this paper, we first show (cf. Theorem 1) that for any set of q non-cooperative K-state robots, there exists a 3-regular graph G, and two pairs $\{u, u'\}$ and $\{v, v'\}$ of neighboring nodes, such that any robot of the set, starting from u or u', fails to traverse the edge $\{v, v'\}$. The graph G has $O(qK)$ nodes, thus improving the $O(K^{O(q)})$ bound of [7] (cf. Corollary 1). By simply plugging this new trap for non-cooperative robots in the trap for cooperative robots by Rollik, we get a new trap of size $\tilde{O}(K^{K^{\cdot^{\cdot^{K}}}})$, with $q+1$ levels of exponential, thus smaller than the one in [7] (cf. Corollary 2).

1.2 Exploration by a Single Robot

Theorem 1 has a significant impact on the space complexity of graph exploration by a single robot. We distinguish the two types of exploration mentioned above *perpetual exploration* and *exploration with stop*, where the robot has to stop once exploration is completed.

[1] The $\tilde{O}$ notation hides logarithmic factors.

In acyclic graphs, exploration with stop is strictly more difficult than perpetual exploration. In particular, it is shown in [4] that exploration with stop in n-node bounded degree trees requires a robot with memory size $\Omega(\log\log\log n)$, whereas perpetual exploration requires $O(1)$ bits.

As mentioned above, when exploration with stop is required, the robot is provided with a pebble. We prove (cf. Theorem 2) that exploration with stop requires a robot with $\Omega(\log n)$ bits for the family of graphs with at most n nodes. Note that, in arbitrary graphs, perpetual exploration and exploration with stop are not comparable because even if perpetual exploration is a simpler task than exploration with stop, in the latter case the robot is given a pebble. Therefore, even if the existence of a trap of at most $K+1$ nodes for any K-state robot described in [6] implies an $\Omega(\log n)$ bits lower bound for the memory size of a robot that performs perpetual exploration in all graphs with at most n nodes, our $\Omega(\log n)$ lower bound is not a consequence of the result in [6].

Finally, we prove (cf. Theorem 3) that there exists an exploration with stop algorithm using a robot with $O(D\log\Delta)$ bits of memory for the exploration with stop of all graphs of diameter at most D and degree at most Δ.

2 Preliminaries

In Section 2.1 we define formally what we mean by a robot exploring a graph. In Section 2.2 we describe the basic properties of a robot. In Section 2.3 we show how to simplify the structure of a robot, for the proofs of the following sections.

2.1 Graphs and Robots

A robot considered in this paper traverses a graph by moving from node to node along the edges of the graph. All nodes are identical and hence indistinguishable to the robot. However, each edge has two labels, each one associated to one of its two endpoints. The labels are arbitrary, except that the edges incident to a node are required to have different labels in their endpoints corresponding to the node. When a robot is in a node, it sees only the labels at the endpoints of the edges incident to the node. This allows the robot to distinguish the edges incident to the node through their unique labels, called *local port numbers.*

An edge may have different port numbers in its two endpoints. When a robot is in a node s and traverses an edge to get to the node t at the other end of the edge, it learns the label at t's endpoint of the edge once it enters t. The robot decides which edge to take to leave t based on this label, as well as on the other local port numbers at t (and hence the degree of t). To compute memory lower bounds, it suffices to consider graphs where both port numbers coincide, and where all nodes have the same degree. In such a graph a robot can be described by a very simple automaton, as we shall see next. Thus, the graphs considered in this paper are *δ-homogeneous* undirected graphs: δ-regular and edge-colored. A graph is *δ-regular* if each of its nodes has degree δ, and it is *edge-colored* if

each edge is labeled with one of the integers in the set $\Delta = \{0, 1, \ldots, \delta - 1\}$ in a way that no two edges incident to the same node have the same color.

When a robot traverses a δ-homogeneous graph, each time it arrives to a node the local environment looks exactly the same as in any other node: all nodes are equal and in each node all local ports are $0, 1, \ldots, \delta - 1$. Thus, the robot decides which edge to take to exit the node based only on its current state. Formally, a *δ-robot* or simply *robot* when δ is understood, is an automaton $A = (\Delta, \mathcal{S}, f, s_0)$, with a finite set of states $\mathcal{S}$, an initial state $s_0 \in \mathcal{S}$, and a transition function $f : \mathcal{S} \to \mathcal{S} \times \Delta$. For a state $s \in \mathcal{S}$ with $f(s) = (s', i)$, denote $f_{st}(s) = s'$ and $f_\ell(s) = i$. The δ-robot A moves on a δ-regular graph as follows. Initially A is placed on a node of the graph in state s_0. If A is in a node v in state s then A moves to the node v' such that the edge $\{v, v'\}$ is labeled $f_\ell(s)$, and changes to state $f_{st}(s)$. We say that A *traversed* the edge $\{v, v'\}$. We assume that every state $s \in \mathcal{S}$ of A is reachable from s_0 (unreachable states do not affect the behavior of A and can be ignored). In Section 4 we will consider an extended definition of a robot that can drop a pebble in a node and pick it up when it returns to the node to drop it somewhere else.

A *trap* for a set of δ-robots is a pair (G, U), where G is a δ-homogeneous graph and U is a set of nodes, such that if all the robots are placed in nodes of $u \in U$, each in its initial state, then there will be an edge $\{v, v'\}$ that is never traversed by the robots.

2.2 Basic Properties

Consider a robot $A = (\Delta, \mathcal{S}, f, s_0)$. The transition function f defines a directed labeled graph $G(A) = (\mathcal{S}, F)$ with node set $\mathcal{S}$ and arc set F, such that the arc $s \to t \in F$ iff $f_{st}(s) = t$, and the arc has label $f_\ell(s)$. Notice that the labeled graph $G(A)$ together with the starting node s_0 completely determine the robot A.

Each node of $G(A)$ has out-degree 1 because f is a function. It follows that $G(A)$ consists of a simple, possibly empty path starting in s_0 and ending in some node s_1, followed by a simple cycle starting and ending in s_1. This is because we assume that A has no unreachable states and $\mathcal{S}$ is finite. Thus, the arc labels of the path define a *path word* W_0 over Δ, $|W_0| \geq 0$, and the arc labels of the cycle define a *cycle word* W over Δ, $|W| \geq 1$. Clearly, $|W_0 W| = |\mathcal{S}|$. The *footprint* of A is $fp(A) = W_0 W^*$. When A is placed on a node of a graph in state s_0, $fp(A)$ is the sequence of labels of edges traversed by A. The next lemma says that once A reaches a node x of the graph in some state s that belongs to the cycle of $G(A)$, the path that A follows in G is a closed path that includes x; moreover, A returns to x in the same state s.

Lemma 1. *Consider a robot A with path and cycle words W_0, W placed in a node of a graph G. Let x be a node reached by A after at least $|W_0|$ steps, and assume A is in state s at this moment. Then A will eventually be back in x in state s.*

Proof. Consider the behavior of A after at least $|W_0|$ steps. The robot A is thus changing from state to state along the cycle of $G(A)$. Since this cycle is finite, and G is also finite, the robot must be twice in the same node in the same state. Assume for contradiction that A is not twice in (x, s), *i.e.* in node x in state s. Then let x' be the first node (after x) for which A is twice in the same state, s'. Consider the path taken by A from the initial node to the first time it is in node x' in state s', and let x_1 be the last node before entering x' in state s' for the first time. Suppose A is in state s_1 at this time. So A is never twice in (x_1, s_1). When A eventually returns to x' in s', the last node visited is x'' in state s''. But since all the states considered in the lemma are in the cycle of $G(A)$ (because A has taken at least $|W_0|$ steps), it must be that $s'' = s_1$. Thus, $f_\ell(s'') = f_\ell(s_1)$, which implies that $x_1 = x''$ (since G is homogeneous). Then A is twice in (x_1, s_1), a contradiction. □

2.3 Reduced Robots

A robot A is *irreducible* if $G(A)$ satisfies two properties: (i) for any two consecutive (distinct) arcs $s \to s_1 \to s_2$, it holds $f_\ell(s) \neq f_\ell(s_1)$, and (ii) for the two arcs with the same end-node $s \to s_1$, $s_2 \to s_1$, it holds $f_\ell(s) \neq f_\ell(s_2)$. We show here how to obtain an irreducible robot $A' = (\Delta, \mathcal{S}', f', s'_0)$ from a robot A. The behavior of A and of A' on a graph will not be exactly the same, but will be related in the sense that the region of a graph traversed by A cannot be much larger than the region traversed by A'.

Let $\bar{G}(A)$ be the undirected graph corresponding to $G(A)$. Then, if A is irreducible and its simple cycle is of length at least 2, then $\bar{G}(A)$ is edge-colored. Roughly speaking, we want the robot to be irreducible to construct a graph based on $\bar{G}(A)$ on which the robot will be moving. Since the constructed graph must be homogeneous, $\bar{G}(A)$ must be homogeneous. Then we can place A at the beginning of the path of $\bar{G}(A)$ and it will never try to go out of $\bar{G}(A)$. To obtain an irreducible robot A' from A we perform a series of reduction steps that modify its transition function and reachable states. When A, A' are placed on the same node of a graph, the path traversed by A' is contained in the the path traversed by A; essentially A' skips some closed walks of A. These reductions are formally defined next.

A *reduction step* is the operation consisting of transforming a robot $A = (\Delta, \mathcal{S}, f, s_0)$ into another robot $A' = (\Delta, \mathcal{S}', f', s'_0)$ where one of the above properties (i) or (ii) is enforced for two arcs, each corresponding to a type-i or type-ii reduction step. The idea is to repeat type-i steps until no more are possible, and hence the robot satisfies property (i), and then if property (ii) is not satisfied, do a single type-ii step to enforce property (ii). Only type-i reductions change the path traversed by the robot.

A *type-i* reduction step is applicable if $G(A)$ has two consecutive distinct arcs $s \to s_1 \to s_2$ with $f_\ell(s) = f_\ell(s_1)$. First, if $s = s_2$ (so the cycle is of length 2 with same labels), A' is obtained from A by letting $f'(s) = (s, i)$, where $i = f_\ell(s)$; and if $s_1 = s_0$ then $s'_0 = s$. For other states $f' = f$. Otherwise, if $s \neq s_2$, it is possible

that s has $0, 1$, or 2 in-neighbors. In each of these cases A' is obtained from A by the following modifications. If s has 0 in-neighbors, then $s = s_0$; let $s'_0 = s_2$. If s has 1 in-neighbor t ($t \neq s_1$), with $f(t) = (s, i)$, then let $f'(t) = (s_2, i)$; If $s = s_0$ then let $s'_0 = s_2$. If s has 2 in-neighbors t_1, t_2, with $f(t_1) = (s, i)$, $f(t_2) = (s, j)$, then $s \neq s_0$; Let $f'(t_1) = (s_2, i)$ and $f'(t_2) = (s_2, j)$. For other states $f' = f$. After doing these modifications, A' is obtained by removing any unreachable states. Notice that for each one of the previous 3 cases at least one unreachable state is removed, namely s. Thus, at most $K - 1$ type-i reductions are possible, starting from a K-state robot.

We will use the following properties of a type-i reduction. Since $f_\ell(s) = f_\ell(s_1) = i$, if the robot is in a node v of the graph in state s, then it moves to v', where $\{v, v'\}$ is colored i, changes to state s_1, and moves back to v, in state s_2. Thus, it is easy to check that a type-i reduction eliminates this v, v', v loop from the path traversed by the robot in the graph, and makes no other changes to the path; that is, if the path arrives to v from w and then proceeds to w' after traversing the v, v', v loop, after the type-i reduction the robot will go from w to v and then directly to w'. Therefore, before the reduction step, the robot explores a node at most distance 1 from the nodes explored by the robot after the reduction.

Once a type-i reduction step is not applicable in $G(A)$, a single *type-ii* reduction step is used, defined as follows. Assume there are two states such that $f(s) = f(s_1)$, that is, $G(A)$ has two arcs with the same end-node $s \to t$, $s_1 \to t$, and $f_\ell(s) = f_\ell(s_1)$; otherwise the reduction does nothing. Exactly one of s, s_1 must be in the cycle of $G(A)$, let's say s_1. So there is a path from t to s_1. Notice that this path is of length at least 1 (i.e. the cycle of $G(A)$ is of length at least 2), because otherwise $t = s_1$ and there is a loop from t to itself labeled $f_\ell(s)$, and a type-i reduction is applicable.

Recall that $fp(A) = W_0 W^*$. Let W' be the longest common postfix of W_0 and W; $|W'| > 0$ by the type-ii assumption. We consider two cases: $|W_0| > |W'|$ and $|W_0| = |W'|$. In the first case W is a postfix of W_0; let t_1 be the in-neighbor of the node just before W' starts in the simple path of $G(A)$ and let t_2 be the node just before W' starts in the cycle of $G(A)$; thus $f_\ell(f_{st}(t_1)) = f_\ell(t_2)$ is the first letter of W'. In both cases A' is obtained from A by the following modifications: If $|W_0| > |W'|$, let $f'(t_1) = (t_2, f_\ell(t_1))$. If $|W_0| = |W'|$ let $s'_0 = t_2$, and removing any unreachable states.

We use the following two properties of a type-ii reduction. A type-ii reduction does not change at all the path traversed by the robot in the graph. After a type-ii reduction is executed property (ii) is satisfied, and property (i) is not violated.

The previous arguments imply:

Lemma 2. *For any robot $A = (\Delta, \mathcal{S}, f, s_0)$ the robot A' obtained through the longest possible sequence of type-i reductions followed by a type-ii reduction is irreducible. Let k be the number of type-i reduction steps in this sequence. Then $k \leq |\mathcal{S}| - 1$. Assume both start at some node of a given graph. Then any edge traversed by A is at distance at most k from some edge traversed by A'.*

3 A Trap for a Team of Non-cooperative Robots

In this section, we focus on graph exploration by a team of non-cooperative robots.

Theorem 1. *For any set $\mathcal{A}$ of q non-cooperative K-state robots, there exist a 3-homogeneous graph G and two pairs of neighboring nodes $\{u, u'\}$ and $\{v, v'\}$ such that (1) the edge $\{u, u'\}$ is labeled 0, (2) starting at u or at u', any robot in $\mathcal{A}$ fails to traverse the edge $\{v, v'\}$, and (3) G has $10qK + O(q)$ nodes.*

Proof. The proof is by induction on $q \geq 0$. The basic step is $q = 0$. The corresponding graph G is displayed on Figure 1.

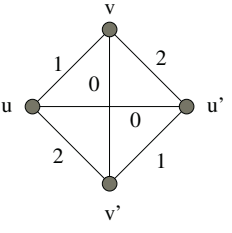


Fig. 1. Basic step of the induction

For the induction step, assume that Theorem 1 holds for q, and let us show that it holds for $q + 1$. Let $\mathcal{A}$ be a set of $q + 1$ non-cooperative K-state robots, and let $A \in \mathcal{A}$. By induction hypothesis, let G_q be an n-node 3-homogeneous graph (where n is $10qK + O(q)$) having two pairs of neighboring nodes $\{u, u'\}$ and $\{v, v'\}$ with the edge $\{u, u'\}$ labeled 0, such that, starting at u or at u', any robot in $\mathcal{A} \setminus \{A\}$ fails to traverse the edge $\{v, v'\}$. We construct a graph G_{q+1} that satisfies Theorem 1 for $\mathcal{A}$.

Let $\widehat{A}$ be an irreducible robot obtained from A as in Lemma 2. Consider its footprint $fp(\widehat{A}) = W_0W^*$, $|W_0W| \leq K$. We concentrate first our attention on $\widehat{A}$, and will come back later to the original robot A. Let us denote by p_i the i-th letter in $fp(\widehat{A})$. Recall that since $\widehat{A}$ is irreducible, its associated undirected graph $\bar{G}(\widehat{A})$ is homogeneous.

Let us place $\widehat{A}$ at node u of G_q, and let us observe its behavior. Let H_1 be the graph obtained from G_q by "cutting" the edge $\{v, v'\}$. In H_1, nodes v and v' are both connected to a pending "half-edge." If $\widehat{A}$ traverses the edge $\{v, v'\}$ in G_q, then in H_1 it traverses one of these two half-edges, say the half-edge e pending at v. We consider two cases, depending on when $\widehat{A}$ traverses e.

Case 1. If $\widehat{A}$ traverses e at step $i \leq |W_0|$ (so p_i is the label of e), then we connect to e a path of length $|W_0| - i$ whose extremity is denoted by w. The edges of this path are labeled $p_{i+1}, \ldots, p_{|W_0|}$. Note that since $\widehat{A}$ is irreducible, two consecutive labels of this path are distinct. At w, we add a ring of length $|W|$. The edges of

this ring are labeled $p_{|W_0|+1}, \ldots, p_{|W_0|+|W|}$ starting and ending at w. Note that since $\widehat{A}$ is irreducible, two consecutive labels of the ring are distinct, and the three labels $p_{|W_0|}, p_{|W_0|+1}$, and $p_{|W_0|+|W|}$ are pairwise distinct. To avoid parallel edges we add a ring of length $2|W|$ at w if $|W| = 2$; to avoid loops when $|W| = 1$, we add a ring with labels *abab*, where a is equal to the single letter of W, and b is different from a and from the label p_i of e.

Case 2. If $\widehat{A}$ traverses e at step $i > |W_0|$, then it traverses e to get into some state s of the cycle in $G(\widehat{A})$; assume this is the j-th state of the cycle (recall that the cycle is assumed to start in the last state of the path of $G(\widehat{A})$). That is, after traversing e, $\widehat{A}$ would traverse edges labeled $p_{|W_0|+j}, p_{|W_0|+j+1}, \ldots$

Let x be the node of H_1 reached by $\widehat{A}$ after $|W_0|$ steps, let W^{-1} be the sequence W written in reverse order, and let $\widehat{A}^{-1}$ be the robot that traverses edges labeled $(W^{-1})^*$. Thus, when $\widehat{A}^{-1}$ starts at x and $\widehat{A}$ reaches x, $\widehat{A}^{-1}$ proceeds as $\widehat{A}$, but backwards. Let $\widehat{A}^*$ be the robot that traverses edges labeled W^*, *i.e.* the robot derived from $\widehat{A}$ by removing states and transitions that involved W_0.

Claim 1. Starting from x, $\widehat{A}^{-1}$ eventually traverses one of the half-edges pending at v or v'.

Proof. Assume for contradiction that $\widehat{A}^{-1}$ does not traverse any of the half-edges pending at v or v'. By Lemma 1, $\widehat{A}^{-1}$ returns to x in the same state, and hence its path in H_1 is a closed path. This path traversed backwards is exactly what $\widehat{A}^*$ traverses from x. So $\widehat{A}^*$ does not traverse any of the half-edges pending at v or v'. Thus, $\widehat{A}$ also does not traverse them, a contradiction. ◇

By Claim 3 we can consider the state reached by $\widehat{A}^{-1}$ after it traverses one of the pending half-edges; assume this is the k-th state of the cycle in $G(\widehat{A})$. We consider two sub-cases, depending on whether $\widehat{A}^{-1}$ traverses the same half-edge as $\widehat{A}$, or not.

Case 2.1. The robot $\widehat{A}^{-1}$ traverses the half-edge e pending at v (*i.e.*, the same as $\widehat{A}$). This implies that the k-th label in W is equal to the $(j-1)$-th label in W, which is the label of e. We consider the section of the cycle of $G(\widehat{A})$ from the j-th state to the k-th state. The end edges of this section have the label of e. We now consider the following word: $W' = W(j-1)W(j)W(j+1)\ldots W(k-1)W(k)W(k+1)\ldots W(j-1)W(j)W(j+1)\ldots W(k-1)W(k)W(k+1)\ldots W(j-1)W(j)W(j+1)\ldots W(k-1)W(k)$ (Note that $W(j-1) = W(k)$ and $|W'| \geq 2 \times |W| + 2$). The two robots $\widehat{A}$ and $\widehat{A}^{-1}$ cannot follow the same path forever after crossing edge e: otherwise, it would mean that moving them both backwards, they would also follow the same path forever (which is impossible since the two robots took different paths at node x in the past). Moreover, the two robots must separate after at most $|W|$ steps, and since $|W'| \geq 2 \times |W| + 2$, they must separate after at least 1 step and at most $|W| - 1$ steps. Now, if the two robots separate from each other at some point after crossing edge e, let us consider the smallest l such that $W(j+l) \neq W(k-1-l)$, *i.e.* the nearest place where the two robots separate from one another. Since $W(j-1) = W(k)$, $l \geq 1$.

By definition of l, we have $W(j+l-1) = W(k-l)$. Since the considered robots are reduced, we also have $W(j+l-1) \neq W(j+l)$. Still by definition of l, we get $W(j+l) \neq W(k-1-l)$. Finally, because we consider reduced robots and we have $W(j+l-1) = W(k-l)$, we get $W(j+l-1) \neq W(k-1-l)$. Overall, this means that $W(j+l-1)$, $W(j+l)$, and $W(k-1-l)$ are pairwise disjoint. We are now ready to construct the following graph: from e, there is a chain that ends in $W(j+l-1)$ at node w, and from this last node a circle W'' goes from $W(j+l)$ to $W(k-l-1)$. Since $W(j+l) \neq W(k-1-l)$ (see above), $|W''| > 2$. When $|W''| > 2$, we add at w a ring of length $|W''|$ labeled W'', starting and ending at w, so that once $\widehat{A}$ and $\widehat{A}^{-1}$ reach w, each one traverses this ring in the opposite direction, and gets back to w in the appropriate state to proceed along the path back to the half-edge e.

Case 2.2. The robot $\widehat{A}^{-1}$ traverses the half-edge e' pending at v' (*i.e.*, not the same as $\widehat{A}$). Suppose when $\widehat{A}^{-1}$ goes through v' it is in state s. We consider again the section of the cycle of $G(\widehat{A})$ from the j-th state to the k-th state (if the section is of length 1, we extend it with W to make sure there is at least one internal node). We connect e and e' by a path with the labels of this section. Thus, when $\mathcal{A}$ traverses the half-edge e, it follows the newly added path, and gets to v' in the appropriate state, namely s, to proceed along the same path of $\widehat{A}^{-1}$ but backwards, and return to x.

In all three cases, every node of degree 2 in the resulting graph is complemented by a pending half-edge, and every node of degree 1 is complemented by two pending half-edges. Every half-edge is labeled consistently so that the resulting graph, denoted by H_2, is 3-homogeneous.

Finally, if $\widehat{A}$ does not traverse any of the two pending half-edges, then we set $H_2 = H_1$. For $l = 0, 1, 2$, let $parity(l)$ be the parity of the number of pending half-edges labeled l in H_2.

Claim 2. For any $l, l' \in \{0, 1, 2\}$, $parity(l) = parity(l')$.

Proof. An edge of H_2 can be considered as two non-pending half-edges. For $l \in \{0, 1, 2\}$, let t_l be the total number of half-edges of H_2 labeled l, and p_l, resp. np_l, be the number of pending, resp. non-pending, half-edges of H_2 labeled l. All nodes in H_2 are exactly of degree 3 and are incident to one half-edge of each label. Thus $t_0 = t_1 = t_2 = |H_2|$ where $|H_2|$ is the number of nodes of H_2. In H_2, if an half-edge is not pending, then it forms an edge with another non-pending edge of H_2 with the same label. Therefore, all the np_l's are even. Since $t_l = p_l + np_l$, t_l and p_l have the same parity, and thus all the p_l's have the same parity. ◇

The parity of the number of pending half-edges of a given label in H_2 is denoted by ϱ. If ϱ is odd, then we add to H_2 a node connected to one of the half-edges, labeled say l, and add two half-edges pending from this node, labeled $l' \neq l$ and $l'' \notin \{l, l'\}$. As a consequence, ϱ becomes even. Now, we pair the half-edges with identical labels. For every pair but one, we connect the two half-edges of the pair by the gadget displayed in Figure 2. (It could be possible to connect

the half-edges by just one edge, but the resulting graph may then not be simple). By labeling the edges of every gadget appropriately, we obtain a 3-homogeneous graph H_3 with only two pending half-edges, of the same label.

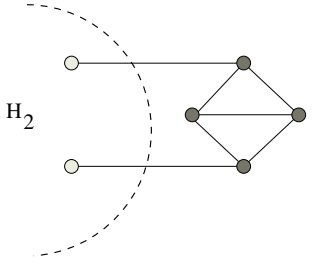


Fig. 2. The gadget for connecting half-edges

Claim 3. Starting from u, $\widehat{A}$ does not traverse any of the two half-edges of H_3.

Proof. By construction, $\widehat{A}$ travels in G_q, and outside G_q it follows the trajectory defined by paths and/or the ring attached to G_q at edges e and e'. Therefore, $\widehat{A}$ does not traverse any of the half-edges of H_2. In particular, it does not traverse any of the two half-edges of H_3. ◇

We define H_4 as the graph obtained from H_3 by adding a "tower" of height $K+1$ connected to the two remaining half-edges, and a gadget closing the tower (see Figure 3):

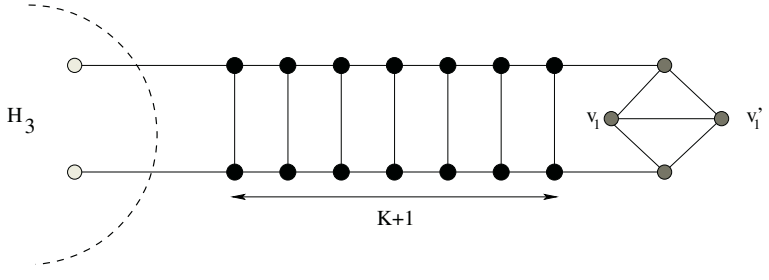


Fig. 3. The "tower" added to H_3

Finally, the two internal nodes of the gadget at the top of the tower in H_4 are denoted by v_1 and v_1' (see Figure 3).

Claim 4. The edge $\{v_1, v_1'\}$ of H_4 is not traversed by A when starting from u.

Proof. By Claim 3, starting from u in H_4, $\widehat{A}$ does not traverse any of the two edges leading from H_3 to the tower. By Lemma 2, the trajectory of A is never

at distance greater than K (where K is the number of states of A) from the trajectory of $\widehat{A}$. Thus, since the tower is of height $K+1$, A never reaches the top of the tower. Therefore, A does not traverse the edge $\{v_1, v_1'\}$. $\diamond$

We repeat the same construction by considering the robot $\widehat{A}$ launched from u' in H_4. More precisely, we construct G_{q+1} from H_4 in the same way H_4 was constructed from G_q. In particular, there is a tower in G_{q+1}, and we define the nodes v_2 and v_2' of G_{q+1} as the two internal nodes of the gadget at the top of this tower. By construction G_{q+1} is 3-homogeneous.

Claim 5. Any robot in $\mathcal{A}$ fails to traverse the edge $\{v_2, v_2'\}$ of G_{q+1} when starting from u or u'.

Proof. By induction hypothesis, starting from u or u', a robot in $\mathcal{A} \setminus \{A\}$ never traverses v, v' in G_q and so will never traverse any of the edges added to obtain G_{q+1}, and hence does not traverse the edge $\{v_2, v_2'\}$ of G_{q+1}. From Claim 3, starting from u, A fails to traverse the edge $\{v_1, v_1'\}$ of H_4. This edge being the one that is "opened" to construct G_{q+1} from H_4, A fails to reach any of the two nodes v_2 or v_2' in G_{q+1}. Finally, by construction of G_{q+1} from H_4, A fails to reach any of the two nodes v_2 or v_2' in G_{q+1} when starting from u', in the same way A fails to reach any of the two nodes w or w' in H_4 when starting from u. $\diamond$

To complete the proof, it just remains to compute the size of G_{q+1}.

Claim 6. $|G_{q+1}| \leq |G_q| + 10K + O(1)$.

Proof. We give simple upper bounds on the size of the intermediate graphs. First, we have $|H_2| \leq |G_q| + K + O(1)$. Moreover, there are at most one half-edge pending from every added node in H_2. For each pair of pending half-edges, we added four nodes, and thus $|H_3| \leq |H_2| + 2K + O(1)$. Finally, the tower has $2K + O(1)$ nodes and thus $|H_4| \leq |G_q| + 5K + O(1)$. The same procedure for the starting node u' contributes to another $5K + O(1)$ additional nodes. The result follows. $\diamond$

As a direct consequence of the previous claim, $|G_{q+1}| \leq 10qK + O(q)$, which completes the proof of Theorem 1. $\square$

By simply rewriting Theorem 1, we derive a bound of the size of the smallest trap for a set of q non-cooperative K-state robots, improving the one by Rollik [7]:

Corollary 1. *For any set of q non-cooperative K-state robots, there exists a trap of size $O(qK)$.*

By simply plugging this latter bound in the construction by Rollik [7] for team of cooperative robots, we get:

Corollary 2. *For any set of q cooperative K-state robots, there exists a trap of size $\tilde{O}(K^{K^{\cdot^{\cdot^{\cdot^{K}}}}})$, with $q+1$ levels of exponential.*

4 Bounds for Exploration with Stop

In this section, we consider the *exploration with stop* problem, in which a robot must traverse all edges of the graph, and eventually stop once this task has been achieved. A robot cannot solve this task in graphs with more nodes than its number of states, by Lemma 1. Thus, the robot is given pebbles that it can drop and take to/from any node in the graph. It is known that any finite robot with a finite source of pebbles cannot explore all graphs [7]. On the other hand, it is known that a robot with unbounded memory can explore all graphs, using only one pebble [5]. An important issue is to bound the size of the robot as a function of the size of the explored graphs.

A *δ-p-robot with a pebble* or simply *p-robot* when δ is understood, is an automaton $A = (\Delta, \mathcal{S}, f, s_0, s_f)$, with a finite set of states $\mathcal{S}$, $s_0, s_f \in \mathcal{S}$, and

$$f : \mathcal{S} \times \{0,1\} \to \mathcal{S} \times \Delta \times \{pick, drop\}.$$

Every state $s \in \mathcal{S}$ has a component $p(s) \in \{0,1\}$ that indicates if A has the pebble, $p(s) = 1$, or not, $p(s) = 0$. For the *initial state,* s_0, $p(s_0) = 0$; for the *stop state,* s_f, $p(s_f) = 1$. Each node v of the graph is in some state $p(v) \in \{0,1\}$ that indicates if the pebble is in v, $p(v) = 1$, or not, $p(v) = 0$. The initial state of the graph satisfies: $p(v) = 1$ for exactly one node v. We will assume the robot is placed initially in the node with the pebble.

The movement of a δ-p-robot A on a δ-regular graph is represented by a sequence of *configurations*, each one consisting of the state of the robot and the state of the graph. For the initial configuration, A is placed on some node of the graph in state s_0, and the pebble is in exactly one node. In general, if A is in a node v in state s in some configuration, we compute $f(s, p(v)) = (s', i, b)$. In the next configuration A will be in the node v' such that the edge $\{v, v'\}$ is colored i, in state s'. Also in the next configuration: if $b = drop$ then $p(v) = 1$ and $p(s') = 0$, and if $b = pick$ then $p(v) = 0$ and $p(s') = 1$. It is assumed that b can be equal to *drop* only if $p(s) = 1$ and b can be equal to *pick* only if $p(v) = 1$.

A robot A *explores with stop* a graph if after starting in any node of the graph that has the pebble, it traverses all its edges and enters a stop state. A graph which A does not explore with stop is called a *trap* for A.

The next theorem shows that a p-robot that performs exploration with stop in all graphs of at most n nodes requires $\Omega(n^{1/3})$ states, or equivalently $\Omega(\log n)$ bits of memory.

Theorem 2. *For any K-state p-robot there exists a trap of size $O(K^3)$.*

Proof. Let $A = (\Delta, \mathcal{S}, f, s_0, s_f)$ be a K-state p-robot. We construct a trap of size $O(K^3)$ for A. For that purpose, we consider the restriction of A to states s such that $p(s) = 0$ and input 0 (on nodes with no pebble). This defines a robot (with no pebble, as in Section 2.1) except that some states may be unreachable from s_0. For every state s of this robot, we consider the robot A_s that has s as

initial state, and includes only reachable states from s. Let $\mathcal{A} = \{A_s\}$ be the set of all these robots. Thus, $|\mathcal{A}| \leq K$.

Let G be a graph satisfying Theorem 1 for the set $\mathcal{A}$. Remove edges $\{u, u'\}$ and $\{v, v'\}$ from G. Consider two copies of the resulting graph, with the four nodes of degree 2 indexed by the index of the copy, 1 and 2. These nodes are re-connected as follows. Let c be the color of the deleted edge $\{v, v'\}$. Create two edges $\{v_1, v'_2\}$ and $\{v'_1, v_2\}$ with color c. The resulting graph is denoted by G_1 (see Figure 4).

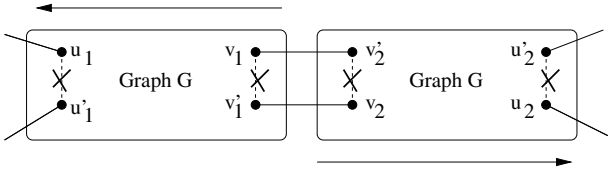


Fig. 4. The graph G_1

Consider an infinite ternary tree modified as follows. Each node is replaced by a 6-cycle. Edges of the cycles are labeled alternatively 1 and 2. Then, edges of the infinite tree are replaced by two "parallel" edges labeled 0, as depicted on Figure 5. The resulting graph is denoted by T.

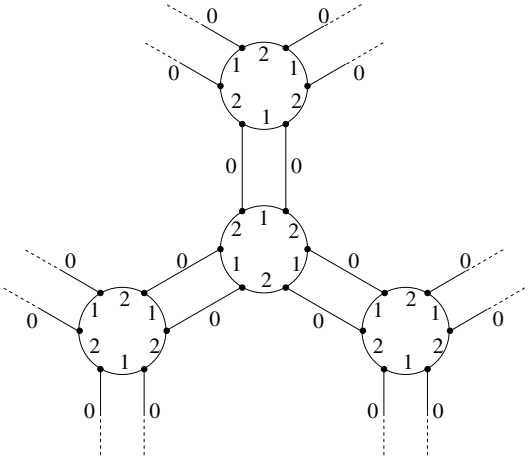


Fig. 5. The modified infinite tree T

The two graphs G_1 and T are composed by replacing every pair $\{\{x, y\}, \{x', y'\}\}$ of parallel edges in T by a copy of G_1. More precisely, x, y, x', y' are respectively connected to nodes u_1, u'_2, u'_1, u_2 in G_1. These new edges are labeled 0. The resulting graph is denoted by G_2. A "meta-edge" of G_2 is defined as a copy of G_1 replacing a parallel edge of T.

By definition of G and $\mathcal{A}$, the p-robot A is unable to traverse a meta-edge of G_2 without the help of the pebble[2]. We now modify G_2 to obtain a graph G_3 such that the p-robot A is unable to explore G_3, even with the pebble. G_3 contains $O(K)$ 6-cycles of T, and thus has at most $O(K^3)$ nodes. The transformation from G_2 to G_3 is technical and very similar to the transformation used in [6] and in [7]. Thus we only sketch the construction of G_3, skipping technical details. Since any p-robot cannot go from a 6-cycle to another 6-node cycle of G_2 without using the pebble, we define *key* steps as those for which the last time the p-robot leaves a 6-cycle with the pebble, go through a meta-edge, and enters another 6-cycle with the pebble. Because the number of states is finite, A will eventually be twice in the same state at these key steps, at two nodes w and w'. With the same technique as in [6], we identify the nodes w and w'. This leads to the graph G_3 with the desired properties, that is G_3 has $O(K)$ 6-cycles, and thus $O(K)$ "parallel" edges. In each pair of "parallel" edges, there is a copy of G_1. Since G_1 has $O(K^2)$ nodes, then G_3 has $O(K^3)$ nodes. □

Theorem 3. *There exists an exploration with stop algorithm which requires $O(D \log \Delta)$ bits of memory when performed in the family of graphs with diameter at most D and degree at most Δ.*

Proof. We describe an algorithm called `DFS-with-stop`, that enables a robot to explore all graphs, with stop. Exploration is achieved by a traversal of the graph similar to DFS. Let u_0 be the initial position of the robot. The pebble is dropped at u_0, and will remain there until exploration is completed. The exploration proceeds in a sequence of phases. At phase $i \geq 1$, the robot performs a DFS at depth i. At any time during each phase the robot keeps in memory the current sequence of port numbers leading back to u_0 in the DFS tree. This takes $O(i \log \Delta)$ bits of memory during phase i, in a graph of maximum degree Δ. At the beginning of Phase i, the robot sets the variable $stop \leftarrow true$. The robot traverses the edges incident to a node u in increasing order of their labels. When the robot leaves the current node u, and enters some node v, it proceeds as follows. If the pebble is at v, then the robot backtracks. Otherwise, if the current depth of the DFS is $\leq i-1$, then the robot carries on the DFS traversal. If the current depth of the DFS is equal to i, then the robot checks whether v has already been visited or not during a previous phase. For that purpose, the robot performs an auxiliary DFS of depth $i-1$ from v. This again requires $O(i \log \Delta)$ bits of memory for storing the sequence of port numbers leading back to v in the auxiliary DFS tree. If the robot finds the pebble during the execution of the auxiliary DFS, then v is at distance $\leq i-1$ from u_0, and thus it has already been explored during a previous phase. If the robot does not find the pebble during the execution of the auxiliary DFS from v, then it sets the variable $stop \leftarrow false$. After completion of the DFS at Phase i, the robot stops if and only if $stop = true$.

[2] Since the $\{u, u'\}$ edges are "open", the proof requires to consider the last time the p-robot is in a u node; this is deferred to the full version of the paper.

Else, it carries on exploration, by starting Phase $i+1$. Clearly, the robot stops after Phase $D+1$ in a graph of diameter D. The memory requirement of this exploration algorithm is dominated by the storage of the sequences of port labels corresponding to two paths (one for the DFS, one for the auxiliary DFS). These paths are of length at most $D+1$ in the family of graphs with diameter D, and thus contributes for $O(D \log \Delta)$ when the degree of the graph is at most Δ. □

5 Conclusions

We have proved that exploration with stop (using one pebble) requires $\Omega(\log n)$ bits for the family of graphs with at most n nodes. In [6], the same lower bound holds for perpetual exploration. In fact, [6] proves that perpetual exploration requires $\Theta(D \log \Delta)$ for the family of graphs of diameter at most D and degree at most Δ. This latter result is obtained by proving that DFS-exploration is space-optimal. We thus ask the following question: is $\Omega(D \log \Delta)$ bits of memory required to explore with stop all graphs of diameter at most D and degree at most Δ?

References

1. M. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation* **176**: 1–21, 2002. Prel. Version in STOC 1998.
2. M. Bender and D. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In 35th Ann. Symp. on Foundations of Computer Science (FOCS), pages 75–85, 1994.
3. L. Budach. Automata and labyrinths. *Math. Nachrichten,* pages 195–282, 1978.
4. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree Exploration with Little Memory. In 13th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA), pages 588–597, 2002.
5. G. Dudek, M. Jenkins, E. Milios, and D. Wilkes. Robotic Exploration as Graph Construction. *IEEE Transaction on Robotics and Automation* **7**(6): 859–865, 1991.
6. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph Exploration by a Finite Automaton. In 29th International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS 3153, pages 451–462, 2004.
7. H.-A. Rollik. Automaten in planaren graphen. *Acta Informatica* **13**: 287–298, 1980.
8. C.-E. Shannon. Presentation of a Maze-Solving Machine. In 8th Conf. of the Josiah Macy Jr. Found. (Cybernetics), pages 173–180, 1951.

Communications in Unknown Networks: Preserving the Secret of Topology

Markus Hinkelmann and Andreas Jakoby

Institut für Theoretische Informatik,
Universität zu Lübeck, Ratzeburger Allee 160, 23538 Lübeck, Germany
{hinkelma, jakoby}@tcs.uni-luebeck.de

Abstract. Cryptography investigates security aspects of data distributed in a network. This kind of security does not protect the secrecy of the network topology against being discovered if some kind of communication has to be established. But there are several scenarios where even the network topology has to be a part of the secret.

In this paper we study the question of communication within a secret network where all processing nodes of the network have only partial knowledge (e.g. given as routing tables) of the complete topology. We introduce a model for measuring the loss of security of the topology when far distance communication takes place. We will investigate lower bounds on the knowledge that can be deduced from the communication string. Several kinds of routing tables are not sufficient to guarantee the secrecy of topology. On the other hand, if a routing table allows to specify the direction from which a message is coming from we can run a protocol solving the all–to–all communication problem such that no processing node can gain additional knowledge about the network.

Finally, we investigate the problem, whether a knowledge base can be generated from local knowledge of the processing nodes without losing the state of secrecy. It will be shown that this is not possible for static networks and most kinds of dynamic networks.

Keywords: communication in unknown networks, security of the topology, network entropy.

1 Introduction

Talking about cryptographic aspects in computer science one usually thinks about some private data that has to be kept as a secret. But this kind of security does not protect a secret communication network against being discovered if some kind of communication takes place. Consider for example the situation of a secret agent in the following scenario:

> The evil secret agent Fat Bastard is spying on Austin Powers. To send his results to Dr. Evil he uses a network of couriers who do not know for whom they are working. To protect the network against attacks of the

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 155–169, 2005.

intelligence service every courier only knows a local part of the courier network. Hence, each single attack only destroys a small part of the network and does only give negligible knowledge about the remaining network. Furthermore, if a local attack occurs all neighbours of the attacked courier may disappear. Every courier has a local instruction set,

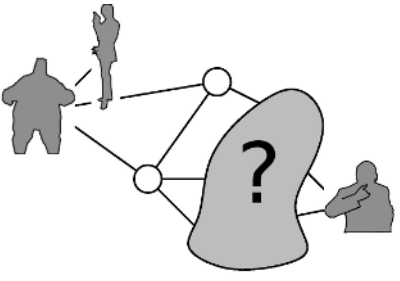


Fig. 1. Fat Bastard spies on Austin Powers

telling him how to forward a message. To improve efficiency one may want to reduce the number of messages received by every member of the network. Note that the intelligence service may get additional knowledge about the topology when analysing the amount of received messages and the local instruction set of a courier.

The fact that a courier has restricted access of the network topology is of strategic importance. If the network is located in an hostile environment it is necessary to reduce the risk of the couriers from being compromised. Furthermore, the existence of central entities in the network has to be avoided [21]. We enhance the security of those networks if the couriers have as little knowledge about the network as possible. If the knowledge about the network is limited, i.e. it is a subject of privacy, then it will also be a hard task for the enemy to choose where to place loyal forces to intercept a transmission.

In this paper we focus on the question whether all–to–all communication is possible if we keep the network topology as a secret.

Communication and Computation in Unknown and Restricted Networks

The interplay of network topology and communication security has been studied extensively in the last decade. So far, one has examined the effects of restricted networks for secure and private computations and communications (see e.g. [3, 10, 11, 14, 18]). The goal is to compute a function depending on the secret inputs of the network nodes (called players) such that after the computation is done no player knows anything about the secrets of the other players that cannot be derived from the function value and its own secret. The authors investigate

the question whether privacy of data can be preserved if the communication network is restricted. The range of possible attacks varies between byzantinean players [9] and honest-but-curious players [2, 8], where the players follow the protocol precisely but are allowed to "gossip" afterwards.

Depending on the computational power of the players we distinguish between cryptographically secure privacy and information theoretically secure privacy. In the first case we assume that no player is able to gain any information about the input bits of the other players within polynomial time [19, 20]. In the second case we do not restrict the computational power of the players. This notion of privacy (sometimes called unconditional privacy) has been introduced by Ben-Or et al. [2] and Chaum et al. [8].

In the papers cited so far, the authors assume that the network is fixed and known in advance. The situation changes when we consider unknown or dynamic networks. This kind of networks is investigated if focusing on ad–hoc networks. The main security attributes that are considered for ad–hoc networks are *availability* (survival against an attack), *confidentiality* (hiding sensitive information), *integrity* (accepted data is not corrupted), *authentication*, and *non-repudiation* (a message cannot be denied by the sender). But only confidentiality addresses privacy concerns [21]. Known protocols that achieve secure communication in an ad–hoc network are usually based on discovering parts of the existing topology at first and using this knowledge to distribute data. Therefore, they are based on the assumption that the topology is stable for a sufficiently long period of time. If Byzantinean players may occur most papers are dedicated to protocols for finding a trusted path from the sender to the receiver [4, 5, 6]. Malicious links can be avoided by assigning high costs, resp. weights, to these links. [1].

All papers cited above ignore the aspect that the network topology might be a secret for itself. Focusing on anonymous communication one gets some kind of a model that considers the topology as a secret [7, 15, 17]. These papers study the traffic analysis problem, i.e. the problem of hiding the sender or the receiver of a message or even the path used for routing a message: Chaum introduces a central entity called *mix* [7]. In his protocol a message is first sent to the mix which forwards the message to the receiver and clouds the relationship of the sender and the receiver. *Onion routing* extends this technique such that proxies in the network establish an anonymous channel between the sender and the receiver [15, 17]. Basically, the sender chooses a delivery path and encodes the path information in the message. Step by step, the proxies decode the successor in the path and forwards the message. The proxies en route work as distributed mixes of the path. Hence, these solutions of the traffic analysis problem explicitly use given routing information in the network.

Our Security Goal

We focus on the problem of all-to-all communication in an unknown network where the topology is the subject of secrecy. Therefore, we introduce a model called *advised network* that combines a communication network and partial knowledge about it. We present a method for measuring the information gain

on the network topology, resp. the loss of network secrecy, when performing a protocol that establishes arbitrary communications between the players. The model of measuring the information gain is closely related to Shannon's entropy function [16] and the mutual information.

Since players which do not know anything about the network — nor even their neighbourhood — cannot perform any kind of useful communication, we make some knowledge $\mathcal{K}$ of the topology available to every player. Then, we will investigate the information gain on the network topology G that can be deduced by examining the communication string C. More precisely, we will investigate the conditional mutual information $\mathrm{I}(G;C|\mathcal{K})$. Moving knowledge about the protocol into the knowledge base one can always guarantee that $\mathrm{I}(G;C|\mathcal{K}) = 0$. Therefore we are also interested in the uncertainty of the topology if the knowledge base is given, i.e. $\mathrm{H}(G|\mathcal{K})$.

The paper is organised as follows. In the following section we will formally introduce our model of advised networks that combines the network topology with partial knowledge. Furthermore, we introduce the network entropy and the corresponding mutual information as a measure for the amount of information given by the communication string about the topology. Section 3 is dedicated to the problem of determining necessary conditions of the knowledge for all–to–all communication. We will show that some knowledge about the distances in the network will always be revealed during the execution of a protocol solving all–to–all communication. In Section 4 we will analyse knowledge bases that correspond to routing tables. If the routing table defines a spanning tree of the network, then partial knowledge of this routing table is sufficient to find a protocol that guarantees $\mathrm{I}(G;C|\mathcal{K}) = 0$. Then, in Section 5 we will investigate the question whether it is possible to construct some kind of routing table–like knowledge base from local network knowledge like the neighbourhood. We will show this is not possible for static networks. Furthermore, we will discuss this problem for dynamic networks. We will show that the construction of a sufficient knowledge base is possible if and only if the players are added to the network node by node. Section 6 will conclude this paper. This work is based on [13].

2 Preliminaries

$G = (V, E)$ denotes a graph, V are the vertices and E are the edges of G. For a graph G the edges of G are given by $\mathrm{E}(G)$ and the vertices by $\mathrm{V}(G)$. We refer $|V(G)|$ to be the size of G and $\mathcal{G}_n$ to the set of graphs with n vertices.

A **(communication) network** is an undirected graph $G = (V, E)$. The edges are called **links**. The vertices are called **processing nodes** (PN for short). The set $\mathrm{nbh}(P_i)$ is called the **neighbourhood** of PN P_i, i.e. the set of all PNs that have links to P_i. A **path** between two PNs $P_i, P_j \in V$ in G is a acyclic subgraph $\boldsymbol{\pi_G(P_i, P_j)}$ with $\mathrm{V}(\pi_G(P_i, P_j)) = \{v_1, v_2, \ldots, v_k\}$, $v_1 = P_i$, $v_k = P_j$, and $\mathrm{E}(\pi_G(P_i, P_j)) = \{\{v_i, v_{i+1}\} \mid i \in [1, k-1]\} \subseteq \mathrm{E}(G)$. The **length** of $\pi_G(P_i, P_j)$ is the number of links in the path. For easier notation we identify $\pi_G(P_i, P_j)$ with its ordered sequence of PNs.

The networks investigated in this paper are synchronous, i.e. the PNs act in rounds. At the beginning of a round the PNs may receive messages from their neighbours. Then, internal computations are performed. At the end of a round the PNs may send messages to their neighbours that are received in the following round. The communication behaviour of the PNs is specified by the used protocol. Usually protocols are based on some knowledge about the underlying network topology. In the following we assume that a protocol can be executed on every network. To provide the PNs with the knowledge necessary to execute the protocol on the specific network we allow the PNs to use a predefined and network specific data base $\mathcal{K}$. A PN P_i does not need to have a global view of the network. It is rather possible that the data base includes knowledge about P_i's neighbourhood, how many PNs exist in the network, and that the network is connected. The data base grants P_i only a restricted access to a local data base $\mathcal{K}_i \in \mathcal{K}$. More formally we define:

Definition 1. *Let* $\boldsymbol{\mathcal{R}}$ *be the set of all local data bases. A mapping* $\boldsymbol{\mathcal{T}}: \mathcal{G}_n \longrightarrow \mathcal{R}^n$ *is called* **knowledge base generator**. *The* **knowledge base of** $\boldsymbol{G}$ *is denoted as* $\boldsymbol{\mathcal{K} := (\mathcal{K}_1, \ldots, \mathcal{K}_n) = \mathcal{T}(G)}$ *and* $\boldsymbol{\mathcal{K}_i}$ *is called* **knowledge base of PN** $\boldsymbol{P_i}$. *The tuple* $\boldsymbol{(G, \mathcal{K})}$ *is called* **advised network**.

Note that a local knowledge base $\mathcal{K}_i$ may not reveal the whole network to PN P_i. Thus, there may exist several networks such that $\mathcal{K}_i$ is the knowledge base provided to P_i. Hence, an additional information source for P_i about the network topology is given by the communication sequence when performing a protocol. Let $\boldsymbol{\mathcal{C}_{\mathcal{P},\mathcal{K},i}}$ be the set of all communication sequences C_i seen by P_i using $\mathcal{K}_i$ when performing a protocol $\mathcal{P}$. If $\mathcal{P}$ is deterministic then $|\mathcal{C}_{\mathcal{P},\mathcal{K},i}| = 1$. To address the possible networks we define

$$\boldsymbol{\mathcal{G}[\mathcal{K}_i]} := \{G \in \mathcal{G}_n \mid \exists \mathcal{K}' :\ \mathcal{T}(G) = \mathcal{K}' = (\mathcal{K}'_1, \ldots, \mathcal{K}'_i, \ldots, \mathcal{K}'_n) \text{ and } \mathcal{K}'_i = \mathcal{K}_i\}.$$

$$\boldsymbol{\mathcal{G}[\mathcal{K}_i, C_i]} := \{G \in \mathcal{G}_n \mid \exists \mathcal{K}' :\ \mathcal{T}(G) = \mathcal{K}' = (\mathcal{K}'_1, \ldots, \mathcal{K}'_i, \ldots, \mathcal{K}'_n) \text{ and } \mathcal{K}'_i = \mathcal{K}_i \text{ and } C_i \in \mathcal{C}_{\mathcal{P},\mathcal{K}',i}\ \}.$$

The purpose of this paper is to find a knowledge base and a protocol that solves a communication problem such that no additional knowledge can be obtained by listening to the received messages, i.e. $\mathcal{G}[\mathcal{K}_i] = \mathcal{G}[\mathcal{K}_i, C_i]$. In particular, we will focus on the **all–to–all communication problem (A2A)**, i.e. for all pairs $(P_i, P_j) \in V \times V$ the PN P_i sends an individual message $m_{i,j}$ to P_j. After receiving $m_{i,j}$ the receiver P_j is able to identify the sender P_i.

We are interested in the amount of uncertainty of the PNs about the network. To measure the uncertainty we adapt *Shannon's* entropy function [16] and the mutual information that are defined as follows. For two discrete random variables X, Y the **probability of** $\boldsymbol{x}$ is denoted as $\boldsymbol{p_x := \mathbb{P}(X = x)}$ and the **conditional probability of** $\boldsymbol{x}$ **given** $\boldsymbol{y}$ is denoted as $\boldsymbol{p_{x|y} := \mathbb{P}(X = x | Y = y)}$. The **entropy of** $\boldsymbol{X}$, resp. the **conditional entropy of** $\boldsymbol{X}$ **given** $\boldsymbol{Y}$, is defined by

$$\mathbf{H}(\boldsymbol{X}) := -\sum\nolimits_x p_x \cdot \log(p_x) \quad \text{and} \quad \mathbf{H}(\boldsymbol{X}|\boldsymbol{Y}) := -\sum\nolimits_y \sum\nolimits_x p_y \cdot p_{x|y} \cdot \log(p_{x|y}).$$

The **conditional mutual information between X and Y given Z** is defined by

$$\mathbf{I(X;Y|Z)} := \mathrm{H}(X|Z) - \mathrm{H}(X|Y,Z).$$

Definition 2. *Let $(G,\mathcal{K})$ be an advised network, let $\mathcal{P}$ be a protocol solving a communication problem on $(G,\mathcal{K})$, and let $P_i \in \mathrm{V}(G)$. The knowledge base $\mathcal{K}_i$ defines the discrete random variable X with values in $\mathcal{G}_n$. Note that $\mathbb{P}(X = G'|\mathcal{K}_i) > 0$ iff $G' \in \mathcal{G}[\mathcal{K}_i]$. The uncertainty about G of P_i is defined by $\mathbf{H(X|\mathcal{K}_i)}$. We call $\mathrm{H}(X|\mathcal{K}_i)$ the* **network entropy according to P_i and $\mathcal{K}$.** *We say $\mathcal{K}$* **covers the mutual information of $\mathcal{P}$** *iff $\mathrm{I}(X;C_i|\mathcal{K}_i\mathcal{P}) = 0$ for all i and $C_i \in \mathcal{C}_{\mathcal{P},\mathcal{K},i}$. The set $\mathbf{cov(\mathcal{P})}$ is defined as the set of all knowledge bases that cover the mutual information of $\mathcal{P}$.*

Note that using this definition we do not restrict the computational power of a PN that tries to gain some knowledge about the network from the communication sequence. For easier notation we write $\mathrm{H}(G|\mathcal{K}_i)$ instead of $\mathrm{H}(X|\mathcal{K}_i)$ and $\mathrm{I}(G;C_i|\mathcal{K}_i\mathcal{P})$ instead of $\mathrm{I}(X;C_i|\mathcal{K}_i\mathcal{P})$. In our model the distribution of X is unknown to P_i. Thus, all networks in $\mathcal{G}[K_i]$ are indistinguishable for P_i. Therefore, $\mathbb{P}(G_i = G'|\mathcal{K}_i) = \frac{1}{|\mathcal{G}[\mathcal{K}_i]|}$ for all $G' \in \mathcal{G}[\mathcal{K}_i]$ and $\mathrm{H}(G|\mathcal{K}_i) = \log(|\mathcal{G}[\mathcal{K}_i]|)$ is the network entropy according to P_i and $\mathcal{K}$.

Our aim is to find a protocol $\mathcal{P}$ and a knowledge base generator $\mathcal{T}$ such that $\mathcal{P}$ solves a given communication problem on $(G,\mathcal{T}(G)) = (G,\mathcal{K})$ for all networks G and

$$\mathcal{K} \in \mathbf{cov(\mathcal{P})}, \text{ i.e. } \mathbf{I(G;C_i|\mathcal{K}_i\mathcal{P}) = 0} \text{ for all } \boldsymbol{i} \text{ and } \boldsymbol{C_i \in \mathcal{C}_{\mathcal{P},\mathcal{K},i}}\ .$$

If G is known to all PNs, then any protocol that solves A2A fulfils this condition. Since we address the network as the subject of secrecy, we want to **maximise** the network entropy of the PNs. Therefore, the knowledge provided to a PN has to be as small as possible.

3 Lower Bounds on the Knowledge to Solve A2A

In the following we want to present some properties that must be fulfilled by a knowledge base to cover the mutual information of a protocol solving A2A.

Inherent Knowledge

The protocol $\mathcal{P}$ has to solve A2A on all networks. Especially, $\mathcal{P}$ solves A2A on $(G,\mathcal{T}(G))$ where G is a tree. Recall that we do not limit the computational power of the PNs. On a tree for any pair P_i, P_j and any PN P_k on the path from P_i to P_j the PN P_k receives all messages sent from P_i to P_j. Thus, P_k is able to extract the same information from these messages as P_j. Since each PN P_j is able to identify the sender P_i of a message $m_{i,j}$ addressed to it, every PN is able to decide which PN is the destination and which is the sender. Therefore, we assume in the following that the sender and receiver is given in the header of a

message. To solve A2A the PN P_j has to be reachable from all other PNs. Hence, the knowledge base generator has to include the knowledge that G is connected, the size of G, the ID i of P_i and all other used IDs in $\mathcal{K}_i$. We call this **inherent knowledge**.

If the knowledge base of an advised network only contains inherent knowledge, then a protocol that solves A2A for all connected networks has to broadcast any message. But if messages are broadcasted in 2–connected components, some PNs gain partial knowledge about these components. Note that this is not deducible from inherent knowledge.

To minimise the partial knowledge about the network, communication is done using a selective broadcast, resp. via selected links. Therefore, we also add the knowledge about the links used by $\mathcal{P}$ to the inherent knowledge. We assume that these links can be addressed by P_i as $e_1, \ldots, e_k$. The use of selected links motivates the use of routing tables as knowledge bases. In Section 4 we will see that the neighbourhood of a PN does not need to be known in detail. We will present a protocol that solves A2A by addressing explicit neighbours. This protocol can easily be transformed into a protocol that only uses incident links without knowledge of the adjacent PNs.

Monotone Offset Function

If messages travel through a synchronous network they are delayed at least one round by every PN they pass. Hence, the round when a message is received depends on the delay, respectively on the number of PNs passed. We will show that there must be some information about the distances in the knowledge base to communicate in the network without revealing knowledge.

Definition 3. *Let $(G, \mathcal{K})$ to be an advised network and $\mathcal{P}$ be a communication protocol that solves A2A on $(G, \mathcal{K})$. A function $\Delta : V \times V \to \mathbb{N}$ is called* **monotone offset function (MOF) for $(G, \mathcal{K})$ with $\mathcal{P}$**, *iff for all processing nodes $P_i \neq P_j \in V$, for all paths $\pi_G(P_i, P_j) = \{v_0, \ldots, v_k\}$ that are used by $\mathcal{P}$, and for all $l \in [0, k-1]$ it holds that $\Delta(P_i, v_l) < \Delta(P_i, v_{l+1})$.*

If a MOF is given, we can construct a protocol that routes the messages along the paths as defined in $\mathcal{P}$. In addition we delay the messages according to the setting of the MOF. Thus, the delay and the upper bound of the distance between two PNs is given by the MOF. Hence, we get:

Lemma 1. *Let $(G, \mathcal{K})$ be an advised network. If a MOF Δ for $(G, \mathcal{K})$ is deducible from the knowledge base $\mathcal{K}$, then there exists a communication protocol $\mathcal{P}$ such that $\mathcal{P}$ solves A2A on $(G, \mathcal{K})$ and the execution of $\mathcal{P}$ does not reveal more information about the distances in G than $\mathcal{K}$.*

Proof. $\mathcal{P}$ is deterministic and works as follows. Let $\pi_G(P_i, P_j)$ be a path from P_i to P_j in G and Δ be a function fulfilling the property of Definition 3 with $\mathrm{V}(\pi_G(P_i, P_j)) = \{v_0, \ldots, v_k\}$. If a message M has to be sent from P_i to P_j, then v_0 delays M for $\Delta(v_0, v_0)$ rounds and sends M to v_1. For $l \in [1, k-1]$ v_l receives

M after $\Delta(v_0, v_l)$ rounds, it delays M for $\Delta(v_0, v_{l+1}) - \Delta(v_0, v_l) - 1 \leq 0$ rounds and sends M to v_{l+1}. The delay of M equals the values of Δ for all pairs of PNs. Thus, the execution does not reveal more information about the distances as given by $\mathcal{K}$. ■

If no MOF is given, then the communication in the network provides more information about the network topology than without sending messages.

Theorem 1. *Let $(G, \mathcal{K})$ be an advised network and $\mathcal{P}$ a communication protocol that solves A2A on $(G, \mathcal{K})$. If it is not possible to deduce a MOF for $(G, \mathcal{K})$ with $\mathcal{P}$ from $\mathcal{K}$, then $\mathcal{K} \notin \text{cov}(\mathcal{P})$.*

To prove this theorem we show that if no MOF is known, then by examining the communication of $\mathcal{P}$ either some networks can be excluded, i.e. $\mathcal{G}[\mathcal{K}_i] \setminus \mathcal{G}[\mathcal{K}_i, C_i] \neq \emptyset$, or a MOF can be constructed. Hence, the knowledge base in an advised network needs to contain consistent information about the distances or the delay of messages in the network.

Proof. Let $d_{i,j} : \mathbb{N} \longrightarrow 2^{\mathcal{G}[\mathcal{K}_j \mathcal{P}]}$ be a mapping from the delay δ of a message travelling in $(G, \mathcal{K})$ from P_i to P_j to the set $\mathcal{G}[\mathcal{K}_j \mathcal{P} \delta]$ that includes all networks of $\mathcal{G}[\mathcal{K}_i \mathcal{P}]$ for those a message sent from P_i to P_j is delayed by δ.

If there exists processing nodes $P_i \neq P_j$ and delays δ_a, δ_b with $d_{i,j}(\delta_a), d_{i,j}(\delta_b) \neq \emptyset$ and $d_{i,j}(\delta_a) \neq d_{i,j}(\delta_b)$, then there exists a network $G' \in (d_{i,j}(\delta_a) \cup d_{i,j}(\delta_b)) \setminus (d_{i,j}(\delta_a) \cap d_{i,j}(\delta_b))$. W.l.o.g. assume $G' \in d_{i,j}(\delta_a)$. Thus, $d_{i,j}(\delta_b) \subset \mathcal{G}[\mathcal{K}_j \mathcal{P}]$ and $\mathrm{I}(G; \delta_b | \mathcal{K}_j \mathcal{P}) > 0$, i.e. $\mathcal{K} \notin \text{cov}(\mathcal{P})$.

If for all processing nodes $P_i \neq P_j$ there exists a delay $\delta_{i,j}$ with $d_{i,j}(k) = \emptyset$ for all $k < \delta$ and $d_{i,j}(k) = d_{i,j}(\delta)$ for all $k \geq \delta$, then the function $\Delta(i, j) = \delta_{i,j}$ is a monotone offset function for $(G, \mathcal{K})$ with $\mathcal{P}$. The property of Definition 3 follows from the fact that with growing distance to the sender the delay must increase strong monotonically. Assume $\mathcal{K} \in \text{cov}(\mathcal{P})$ then Δ is deducible from $\mathcal{K}$ — a contradiction. ■

4 Routing Tables

Types of knowledge bases that include inherent knowledge and sufficient knowledge about distances in the network are routing tables. Furthermore, routing in most networks as well as in the Internet is done using routing tables. Thus, we want to analyse the properties of a routing table such that A2A is possible by using the table and one cannot reveal additional knowledge about the network. For easier notation we assume that every PN knows its neighbourhood. The functions discussed in the following, i.e. the next_G and the last_G function, depend on the neighbour that sends a message. Note that we can easily modify this function such that they depend only on the used links.

If a message is received by a PN, the routing table provides the information to which neighbours the message has to be send next. These neighbours may depend on the sender and the receiver of the message. In the following definition a routing table is represented by the function next_G.

Definition 4. *Let $(G, \mathcal{K})$ be an advised network where $\mathcal{K}$ includes inherent knowledge. $\mathcal{K}$ fulfils the* **weak routing table property (WRTP)** *on G, if for all P_s, P_t, $P_i \in \mathrm{V}(G)$ holds: There exists a function $\mathrm{next}_G : V^3 \to 2^V$ that is computable by P_i using $\mathcal{K}_i$ such that $|\{P_j | P_i \in \mathrm{next}_G(P_j, P_s, P_t)\}| \leq 1$ and for a path $\pi_G(P_s, P_t)$ with $\mathrm{V}(\pi_G(P_s, P_t)) = \{P_0, \ldots, P_k\}$, $P_s = P_0$ and $P_t = P_k$ it holds that $P_{i+1} \in \mathrm{next}_G(P_i, P_s, P_t)$ for all $i < k$.*

$\mathcal{K}$ fulfils the **consistent routing table property (CRTP)** *on G, if $\mathcal{K}$ fulfils WRTP and for the function next_G that is computable by P_i using $\mathcal{K}_i$ under WRTP holds: The network $ST_G(\mathrm{V}(G), E')$ is a spanning tree of G where $E' = \{\{P_i, P_j\} | \exists P_s, P_t : \ P_j \in \mathrm{next}_G(P_i, P_s, P_t)\}$.*

$\mathcal{K}$ fulfils the **strong routing table property (SRTP)** *on G, if $\mathcal{K}$ fulfils CRTP and it holds: For the function next_G computable by WRTP of $\mathcal{K}$ there is a partially defined function $\mathrm{last}_G : V^3 \to V$ that is computable by P_i using $\mathcal{K}_i$ such that for all P_i, P_j, P_s, P_t holds $\mathrm{last}_G(P_i, P_s, P_t) = P_j$ if $P_i \in \mathrm{next}_G(P_j, P_s, P_t)$ and $\mathrm{last}_G(P_i, P_s, P_t) = \perp$ otherwise.*

The WRTP property defines for all sender–receiver pairs a tree connecting both PNs. If the knowledge base fulfils CRTP, then the next_G–function induces a spanning tree of the network. The SRTP property guarantees that the knowledge base provides information about the predecessors, i.e. the inverse of the next_G–function. By a counter–example we can show that:

Theorem 2. *There exist advised networks $(G, \mathcal{K})$ that fulfil CRTP such that for all communication protocols $\mathcal{P}$ that solve A2A on G it holds that $\mathcal{K} \notin \mathrm{cov}(\mathcal{P})$.*

Proof. Assume $\mathcal{P}$ to solve A2A on an advised network $(G, \mathcal{K})$ with $\mathcal{K} \in \mathrm{cov}(\mathcal{P})$. Let G be the network illustrated in Figure 2 and let $\mathcal{K}$ include inherent knowledge and the function next_G but not last_G. As $\mathcal{P}$ solves A2A the node P_s sends a message M to P_t. $\mathcal{K} \in \mathrm{cov}(\mathcal{P})$ fulfils WRTP, thus, P_1 receives M exactly once, otherwise P_1 learns that P_1, P_2, P_3 belong to the same 2–connected component of G. P_1 gets to know its preceding processing node if P_s sends a message to P_t. Now assume, that every pair of processing nodes exchange messages. Thus, for all sending and receiving processing node each processing node is able to compute the predecessor. If $\mathcal{K} \in \mathrm{cov}(\mathcal{P})$ all communication that is seen by $P_i \in \mathrm{V}(G)$ must be computable from $\mathcal{K}_i$ and the knowledge about $\mathcal{P}$. Hence, last_G is computable from $\mathcal{K}$ and $\mathcal{K}$ fulfils SRTP, a contradiction. ■

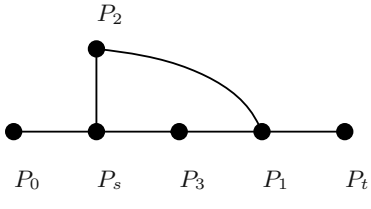


Fig. 2. Network from the proof of Theorem 2

If $\mathcal{K}$ fulfils SRTP the function last_G provides each PN P_i with the size, i.e. the number of nodes of the subtrees corresponding to each neighbour P_j in $\mathrm{nbh}(P_i)$:

$$\mathrm{size}_{\mathrm{ST}}(P_i, P_j) := \textstyle\sum_{P_t \in \mathrm{V}(G)} |\{P_s \mid \mathrm{last}_G(P_i, P_s, P_t) = P_j\}| \ .$$

Note that we can easily compute the function $\text{size}_{\text{ST}}(P_i, P_j)$ of a pair P_i, P_j even if $P_j \notin \text{nbh}(P_i)$ such that $\text{size}_{\text{ST}}(P_i, P_j)$ is the number of nodes reachable from P_j in ST_G without passing P_i.

Theorem 3. *Let $(G, \mathcal{K})$ be an advised network where $\mathcal{K}$ fulfils SRTP. Then, there exists a protocol $\mathcal{P}$ that solves A2A with $\mathcal{K} \in \text{cov}(\mathcal{P})$.*

The protocol routes the messages using next_G and last_G. If a PN P_i has to forward a message M due to next_G, P_i delays M according to the sizes of the subtree including P_i as seen from the receiving neighbours. One can show that P_i does not get further information about the network, especially, P_i gets no additional information about the distances between two nodes. Algorithm 1 describes a protocol that realises one–to–one communication. The procedure can be easily modified to solve A2A.

Algorithm 1 Sending a message M from P_s to P_t
Input: message M, sending PN P_s, receiving PN P_t, next_G, last_G

for all $P_i \in \text{V}(G)$ **do in parallel**
  if $P_i \neq P_s$ **then** receive M **end if**
  if $P_i = P_s$ **then**
    for all $P_j \in \text{next}_G(P_i, P_s, P_t)$ **do in parallel**
      $d_{P_j} \leftarrow \sum_{P_k \in \text{nbh}(P_i) \setminus \{P_j\}} \text{size}_{\text{ST}}(P_i, P_k)$
    end for
  else if $P_i \notin \{P_s, P_t\}$ **then**
    for all $P_j \in \text{next}_G(P_i, P_s, P_t)$ **do in parallel**
      $d_{P_j} \leftarrow -\text{size}_{\text{ST}}(P_i, \text{last}_G(P_i, P_s, P_t)) + \sum_{P_k \in \text{nbh}(P_i) \setminus \{P_j\}} \text{size}_{\text{ST}}(P_i, P_k)$
    end for
  end if
  if $P_i \neq P_t$ **then** delay M for d_{P_j} rounds, send M to P_j **end if**
end for

5 Generating Routing Table–Like Knowledge Bases

So far, we have analysed advised networks with given knowledge bases. When real communication networks are set up a PN gets the knowledge about its environment from a higher authority: The network administrator plugs in the wires, the DHCP–Server answers a broadcast and informs the node about its IP–address and the gateway to the Internet. We want to keep the network topology secret. Thus, a higher authority with a global view on the network should be avoided. Now, we are going to analyse whether the PNs are able to build up SRTP fulfilling routing tables from inherent knowledge and the knowledge about their neighbourhood by exchanging some messages. Recall that the communication sequence of a PN must not give more knowledge about the network than is deducible from the constructed knowledge base.

The networks considered so far have been **static**. In the following we focus on networks that are **dynamic**. These are inspired by ad–hoc networks. We concentrate on dynamic networks where single PNs or networks of PNs are added to a given network. This means, single PNs or communication networks may appear and be connected to the existing network by adding links. The connection is done by updating the nbh–function in the knowledge bases and combining those. More formally, in the beginning a PN P_i uses $\mathcal{K}_i^0$. In the case that something occasionally changes, some of the PNs get updated knowledge bases. Thus, at round T, PN P_i is allowed to use all knowledge bases in $\{\mathcal{K}_i^j \mid j \in [0,T]\}$. Therefore, we assume that $\mathcal{K}_i^T$ includes the contents of all prior knowledge bases. We deal with the question whether the routing tables $\mathcal{K}_i^{T+1}$ can be generated without revealing more knowledge about the network than given by $\mathcal{K}_i^{T+1}$ by analysing the communication that occurs during the generation of the modified routing tables. Since we will discuss the update problem in the following, we will omit the superscript T.

If the nbh–function is updated, the PNs do not have any information about the settings of the SRTP conforming functions next_G and last_G of the added PNs. Thus, for generating a SRTP fulfilling knowledge base of the combined network communication in the network is necessary. In the following we will show that adding single nodes to a network can be handled without revealing more knowledge by the communication than one can deduce from the resulting knowledge base.

Definition 5. *Let $(G',\mathcal{K}')$ and $(G'',\mathcal{K}'')$ be two node disjoint advised networks. $(G,\mathcal{K})$ is called the* **combined network** *of $(G',\mathcal{K}')$ and $(G'',\mathcal{K}'')$ iff $G = (\mathrm{V}(G') \cup \mathrm{V}(G''), \mathrm{E}(G') \cup \mathrm{E}(G'') \cup E)$ for some links $E \subseteq \{\{P_i,P_j\} | P_i \in \mathrm{V}(G'), P_j \in \mathrm{V}(G'')\}$. For each PN $P_i \in \mathrm{V}(G')$ the knowledge base $\mathcal{K}_i$ consists of $\mathcal{K}_i'$, the size of G, and the modified neighbourhood function of P_i in G. Analogously, the knowledge base for a processing node in $\mathrm{V}(G'')$ is modified.*

If we add a single PN to an existing network, the new PN can select one of its neighbours to be its exclusive neighbour in the next_G–tree. Using a protocol analogously to Algorithm 1 we can update the knowledge bases of the remaining PNs. Hence, we get:

Lemma 2. *Let $(G',\mathcal{K}')$ and $(G'',\mathcal{K}'')$ be two disjoint advised networks where $\mathcal{K}',\mathcal{K}''$ fulfil SRTP. If the network G'' consists of a single PN, then there exists a protocol $\mathcal{P}$ that generates the knowledge base of the combined network $(G,\mathcal{K})$ such that $\mathcal{K}$ fulfils SRTP and $\mathcal{K} \in \mathrm{cov}(\mathcal{P})$ and no further knowledge can be deduced by any PN performing $\mathcal{P}$.*

Combining Networks

Let $(G',\mathcal{K}')$ and $(G'',\mathcal{K}'')$ be two advised networks both of size at least two as illustrated in Figure 3. Assume that $\mathcal{K}'$ and $\mathcal{K}''$ fulfil the strong routing table property. Now we combine G' and G'' to a connected network G by adding some links. Let $\mathcal{K}$ be the resulting knowledge base. In the following we will focus on

the question whether $\mathcal{K}$ fulfils the strong routing table property, i.e. focusing on the question whether we can construct a next_G–function as used in the definition of the strong routing table property from the local knowledge about $\mathcal{K}', \mathcal{K}''$ and the updated neighbourhoods.

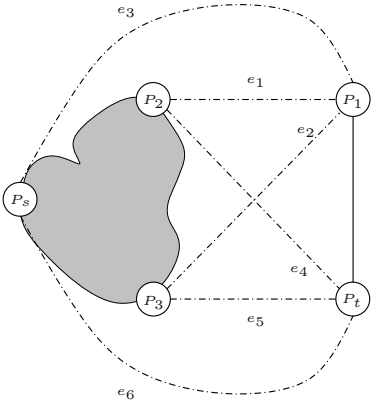


Fig. 3. Possible connections between two subnetworks

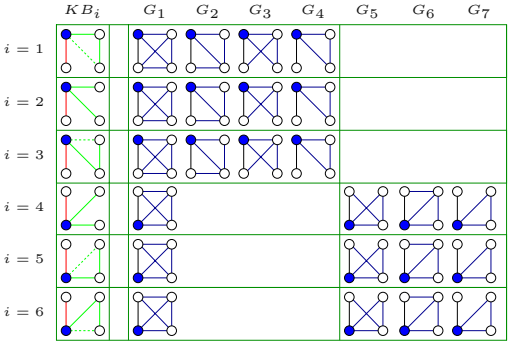


Fig. 4. Knowledge bases and additional knowledge of P_1, resp. P_2 , (first column) and corresponding networks after combining two networks

Theorem 4. *There does not exist a communication protocol $\mathcal{P}$ such that for all sequences of advised networks $(G', \mathcal{K}'), (G'', \mathcal{K}''), \ldots$ where all knowledge bases fulfil SRTP, $\mathcal{P}$ generates a SRTP fulfilling knowledge $\mathcal{K}$ base of the combined network and $\mathcal{K} \in \text{cov}(\mathcal{P})$ where $\mathcal{K}$ only consists of the knowledge given by $\mathcal{K}', \mathcal{K}'', \ldots,$ the updated neighbourhood function and a new* next_G*–function.*

Proof. We consider the case of combining two advised networks $(G', \mathcal{K}')$ and $(G'', \mathcal{K}'')$. $(G, \mathcal{K})$ denotes the combined network. G' and G'' each consist of two PNs. Figure 4 illustrates some examples for $(G, \mathcal{K})$. In the images G' consists on the two nodes on the left hand side and G'' consists of the remaining nodes. The PNs of G' are denoted as P_1 and P_2 where P_1 is drawn in the upper left corner. The graph in the first column gives possible shapes of the knowledge base in the combined network $(G, \mathcal{K})$ for the distinguished PN that is denoted by the solid dot. We denote the knowledge base in row i as KB_i. In remaining columns we give all networks G that may occur for the knowledge base in the same row. The first three rows consider P_1 as the distinguished PN and the last three rows consider P_2. A solid line in the knowledge base KB_i means that the distinguished PN knows that the corresponding link is part of the spanning tree ST_G induced by the next_G–function of $\mathcal{K}$. If the line is dotted, the link exists but is not part of ST_G. In total, there are 15 possibilities for G. But restricting to the knowledge bases in the first column only the networks $G_1, \ldots, G_7$ can occur.

Assume that the knowledge bases $\mathcal{K}'$ does not include any information about G'' and $\mathcal{K}''$ does not include any information about G'. Let $\mathcal{P}$ be a communication protocol that generates the knowledge base of the combined network $(G, \mathcal{K})$ such

that $\mathcal{K}$ fulfils SRTP. Assume that $\mathcal{K} \in \text{cov}(\mathcal{P})$. Since $\mathcal{P}$ and $\mathcal{K}'$ are fixed we omit them in the probabilities. Assume that $G = G_4$. Then, either KB_1, KB_2, or KB_3 has be the resulting knowledge base of P_1 after performing $\mathcal{P}$. Thus, there exists $j \in \{1, 2, 3\}$ with $\mathbb{P}(KB_j) > 0$. Since $\mathcal{K} \in \text{cov}(\mathcal{P})$, the PN cannot distinguish between possible networks. Hence, for all $i \in [1, 4]$ it holds that

$$\mathbb{P}(G_i|KB_j) = \tfrac{1}{4} \qquad \text{and} \qquad \mathbb{P}(G_i, KB_j) = \tfrac{1}{4} \cdot \mathbb{P}(KB_j).$$

In total, there exist 15 networks G after the combination process. Therefore,

$$\mathbb{P}(G_i) = \tfrac{1}{15} \qquad \text{and} \qquad \mathbb{P}(G_i, KB_j) = \tfrac{1}{15} \cdot \mathbb{P}(KB_j|G_i).$$

Thus, $\mathbb{P}(KB_j|G_i)$ can be described independently of G_i for $i \in [1, 4]$:

$$\mathbb{P}(KB_j|G_i) = \tfrac{15}{4} \cdot \mathbb{P}(KB_j).$$

This implies that for $i, l \in [1, 4]$ and $j \in \{1, 2, 3\}$ it holds that

$$\mathbb{P}(KB_j|G_i) = \mathbb{P}(KB_j|G_l).$$

Recall that if G_4 occurs, then only KB_1, KB_2, or KB_3 may be the result of $\mathcal{P}$ for P_1. Thus,

$$\sum_{j\in\{1,2,3\}} \mathbb{P}(KB_j|G_4) = 1 \quad \text{and therefore,} \sum_{j\in\{1,2,3\}} \mathbb{P}(KB_j|G_1) = 1.$$

Especially, it holds that

$$\mathbb{P}(KB_j|G_4) \;=\; \mathbb{P}(KB_j|G_1) \;=\; 0$$

for $j \in \{4, 5, 6\}$. Analogously to our analysis above, we can show for P_2 that KB_4, KB_5, KB_6 are the only possible knowledge bases for P_2 if G_7 occurs. Hence,

$$\sum_i \mathbb{P}(KB_i|G_7) \;=\; \sum_{i\in\{4,5,6\}} \mathbb{P}(KB_i|G_7) \;=\; 1$$

and

$$\mathbb{P}(KB_j|G_4) \;=\; \mathbb{P}(KB_j|G_1) \;=\; 0$$

for all $j \notin \{4, 5, 6\}$ – a contradiction. We can conclude that $\mathcal{K} \notin \text{cov}(\mathcal{P})$. ■

If we restrict the knowledge bases to contain only the inherent knowledge, next_G, and last_G, then it follows by the theorems above:

Corollary 1. *Let $\mathcal{K}$ be a knowledge base fulfilling SRTP. Then, there does not exist a protocol $\mathcal{P}$ such that $\mathcal{K} \in \text{cov}(\mathcal{P})$ and $\mathcal{P}$ generates $\mathcal{K}$ from the inherent knowledge in dynamic networks except the case PNs are added one by one and $\mathcal{K}$ fulfils SRTP.*

Static Networks

Static networks are fixed from the beginning. They never change their shape. In contrast to dynamic networks a protocol $\mathcal{P}$, that generates a SRTP fulfilling knowledge base, cannot resort to the addition of one PN like in Lemma 2. Static networks lack to know the sequence in which the PNs are included into the network.

The question whether knowledge bases can be generated out of inherent knowledge in the static case also influences the dynamic case. When a communication network is set up it usually consists of several PNs. Thus, most dynamic networks originate from a static basis. Such a basis has as well to be provided with a knowledge base that allows all–to–all communication.

Calculating the network entropy we can show:

Theorem 5. *There does not exist a communication protocol $\mathcal{P}$ such that for each communication network G the protocol $\mathcal{P}$ generates a SRTP fulfilling knowledge base $\mathcal{K} \in \mathrm{cov}(\mathcal{P})$ such that only the inherent knowledge and the SRTP* next_G*–/* last_G*–functions are deducible from $\mathcal{K}$.*

To prove this theorem we analyse the network entropy of the network given in Figure 3. One can show that either P_1 learns something about the remaining links incident to P_t or P_t learns something about the remaining links incident to P_1.

6 Conclusions and Open Problems

In this paper we have investigated the secrecy of network topology. Therefore, we have introduced the model of *advised networks*. The available knowledge of the PNs about the network is represented by their knowledge bases. A information theoretic metric called network entropy is introduced. We have investigated some knowledge bases with routing table properties and studied whether these are sufficient to ensure privacy of the network. For knowledge bases that fulfil the strong routing table property a protocol is presented that solves the all–to–all communication problem. This protocol does not reveal additional information about the network topology. Finally, we have dealt with the question whether the knowledge bases can be generated from inherent knowledge. More precisely, we have studied whether a routing table that only consists of SRTP functions can be computed if two or more dynamic networks are combined. We have shown that there does not exist a protocol that generates the desired knowledge bases without giving away more information about the topology than allowed. The only exception is the case where single nodes are added in dynamic environments.

Many problems remain open. We want to minimise the knowledge given to the PNs. Do routing tables provide minimal knowledge? And if not, can we find a knowledge base such that the network entropy is maximal? The dynamic networks used are inspired by ad–hoc networks. We allowed that PNs and links are added. But in ad–hoc networks links and PNs may disappear or some links

are inoperable from time to time. What property is sufficient for a knowledge base in these environments? It also remains as an open problem whether asynchrony increases the network uncertainty and therefore, the security.

Acknowledgements. We would like to thank an unknown referee for useful comments and R. Reischuk for fruitful discussions.

References

[1] B. Awerbuch, D. Holmer, C. Nita-Rotaru, H. Rubens, *An On-Demand Secure Routing Protocol Resilient to Byzantine Failures*, ACM Workshop WiSe, 2002, 21–30.
[2] M. Ben-Or, S. Goldwasser, A. Wigderson, *Completeness Theorem for Non cryptographic Fault-tolerant Distributed Computing,* STOC, 1988, 1–10.
[3] M. Bläser, A. Jakoby, M. Liśkiewicz, B. Siebert, *Private Computation – k-connected versus 1-connected Networks,* CRYPTO, 2002, 194–209.
[4] M. Burmester, Y. Desmedt, *Secure Communication in an Unknown Network Using Certificates,* ASIACRYPT, 1999, 274–287.
[5] M. Burmester, T. Van Le, *Secure Multipath Communication in Mobile Ad hoc Networks,* ITCC, 2004, 405–409.
[6] M. Burmester, T. Van Le, A. Yasinsac, *Weathering the Storm: Managing Redundancy and Security in Ad Hoc Networks,* ADHOC-NOW, 2004, 96–107.
[7] D. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms*, Communications of the ACS, 4(2), 1981, 84–88.
[8] D. Chaum, C. Crépau, I. Damgård, *Multiparty unconditionally secure protocols,* STOC, 1988, 11–19.
[9] D. Dolev *The Byzantine generals strike again*, J. of Algorithms, 3(1), 1982, 14-30.
[10] M. Franklin, N. Wright, *Secure communication in minimal connectivity models*, EUROCRYPT, 1998, 346–360.
[11] M. Franklin, M. Yung, *Secure hypergraphs: privacy from partial broadcast (Extended Abstract),* STOC, 1995, 36–44.
[12] D. M. Goldschlag, M. G. Reed, P. F. Syverson, *Hiding Routing Information*, Information Hiding, 1996, 137-150.
[13] M. Hinkelmann, *Preserving the Secret of Topology*, Diplomarbeit, Institut für Theoretische Informatik, Universität zu Lübeck, 2004
[14] A. Jakoby, M. Liśkiewicz, R. Reischuk, *Private Computations in Networks: Topology versus Randomness,* STACS, 2003, 121–132.
[15] M. Reiter, A. Rubin, *Crowds: anonymity for Web transactions*, ACM Transactions on Information and System Security, 1(1), 1998, 66-92.
[16] C.E. Shannon, *A Mathematical Theory of Communication,* The Bell System Technical Journal, vol. 27, 1948, 379–423 and 623–656.
[17] P. F. Syverson, D. M. Goldschlag, M. G. Reed, *Anonymous Connections and Onion Routing*, IEEE Symposium on Security and Privacy, 1997, 4–7.
[18] Y. Wang, Y. Desmedt, *Secure communication in broadcast channels: the answer to Franklin and Wright's question*, EUROCRYPT, 1999, 446–458.
[19] A. C. Yao, *Protocols for Secure Computations,* FOCS, 1982, 160–164.
[20] A. C. Yao, *How to generate and exchange secrets,* FOCS, 1986, 162–167.
[21] L. Zhou, Z. J. Haas, *Securing Ad Hoc Networks*, IEEE Network, 13(6), 1999, 24–30.

An Improved Algorithm for Adaptive Condition-Based Consensus

Taisuke Izumi and Toshimitsu Masuzawa

Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, 560-8531, Japan
{t-izumi, masuzawa}@ist.osaka-u.ac.jp

Abstract. *Condition-Based Approach* studies restrictions on the inputs of a distributed problem, called *conditions*, to circumvent several impossibility results. Especially, for the synchronous consensus problem, the relation between conditions and time complexity bounds has been studied. In our previous work [12], we introduced the adaptiveness on time complexity of the condition-based approach, and established the *adaptive* condition-based approach: It classifies all possible input vectors into the hierarchical sequence of conditions according to their difficulty called *legality level*. For such hierarchy, adaptive algorithms achieve time complexity depending on the legality level of input vectors. In this paper, we propose an improved version of the adaptive condition-based algorithms for synchronous consensus that achieves better time complexity than the previous one. On the assumption that majority of processes are correct, the proposed algorithm terminates within $\min\{f+2, t+1\}-l$ rounds if $l < f$, where f and t is the actual and the maximum numbers of faults respectively, and l is the legality level of input vectors. Moreover, the algorithm terminates in 1 round if $l \geq t$ and $f = 0$, and terminates within 2 rounds if $l \geq f$ holds. Compared with our previous algorithm, the proposed algorithm improves time complexity by one round in the case of $f = t$ and $l > f$.

1 Introduction

The *consensus* problem is a fundamental and important problem for designing fault-tolerant distributed systems. Informally, the consensus problem is defined as follows: each process proposes a value, and all non-faulty processes have to agree on a common value that is proposed by a process. The (uniform) consensus problem has many applications, e.g., atomic broadcast [3][10], shared objects [1][11], weak atomic commitment [9] and so on. However, despite of the variety of its applications, the consensus problem is known to be unsolvable by deterministic solutions in asynchronous systems subject to only a single crash fault [8]. Thus, several approaches to circumvent this impossibility, such as synchrony [5][7], randomization [2] and unreliable failure detectors [3], have been proposed.

As one of such approaches, the *condition-based approach* is recently introduced [14]. The principle of this approach is to restrict inputs so that the

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 170–184, 2005.

generally-unsolvable problem can become solvable. A *condition* represents some restriction to inputs. In the case of the consensus problem, the condition is defined as a subset of all possible *input vectors* whose entries correspond to the proposal of each process. The first result of the condition-based approach clarifies the condition for which the uniform consensus can be solved in asynchronous systems subject to crash faults [14]. More precisely, this result presented a class of conditions, called *d-legal conditions*, and proved that the d-legal conditions is the class of necessary and sufficient conditions that make the (uniform) consensus solvable in asynchronous systems where at most d processes can crash. Several succeeding researches study application of the condition-based approach to synchronous systems [15][16][19]. Since it is well-known that the uniform consensus problem can be solved in synchronous systems with crash faults, these researches focus on improvement of the time complexity for the restricted input vectors. While it is known that any synchronous uniform consensus algorithm needs at least $\min\{f+2, t+1\}$ rounds for termination, where f and t is the actual and the maximum numbers of faults respectively [4][6], more efficient algorithms can be realized if the condition-based approach is introduced. For example, the algorithm proposed in [16] can solve the uniform consensus within $\min\{f+2, t+1-d\}$ rounds if the input vector is in some d-legal condition, and terminates within $\min\{f+2, t+1\}$ rounds otherwise.

In our previous paper [12], we advanced the condition-based approach, and introduced the notion of adaptiveness on the time complexity of the condition-based approach. Intuitively, the adaptive condition-based approach classifies all possible input vectors into a *legal condition sequence* that is hierarchical division of all possible input vectors according to their difficulty. Execution time of the adaptive condition-based algorithms for a given input vector depends on the location of the input vector in the hierarchy, called *legality level* of the input vector[1]. To explain the legality level more precisely, we present an example for the d-legal condition $C_d^{\max}$: The condition $C_d^{\max}$ consists of the vectors in which the largest value in the vector appears at more than d entries. Clearly, from the definition, $C_d^{\max}$ includes $C_{d+1}^{\max}$. Thus, we can classify all possible input vectors into the hierarchy $C_1^{\max} \supseteq C_2^{\max} \supseteq \cdots C_n^{\max}$. The legality level of an input vector is defined as the location in the hierarchy. For example, the legality levels of two vectors $I_1 =< 0, 1, 1, 1, 1 >$ and $I_2 =< 0, 1, 2, 2, 2 >$ are 3 and 2 respectively. For any input vector with legality level l, the algorithm proposed in our previous result [12] terminates within $\min\{f+2-l, t+1\}$ rounds if $l < f$ holds, within 2 rounds if $l \geq f$ holds, and within 1 round if $f = 0$ and $l \geq t$ holds on the assumption that majority of processes are correct.

In this paper, we improve the time complexity of the adaptive condition-based algorithm. Same as our previous result, the proposed algorithm also assumes that majority of processes are correct. For any input vector with legality level l, it terminates within $\min\{f+2, t+1\} - l$ rounds if $l < f$ holds, within 2 rounds if

[1] The notion of the legal condition sequence and the legality level is similar to that of the *hierarchy* and the *degree* proposed in [17].

$l \geq f$ holds, and within 1 round if $f = 0$ and $l \geq t$ holds. This algorithm achieves the strictly better time complexity than the previous one: It improves the time complexity by one round in the case of $f = t$ and $l < f$. We derive the key idea of this improvement from the scheme proposed in [16]. The main contribution of this paper is to modify this scheme to an adaptive version.

The paper is organized as follows: Section 2 provides the system model, the definition of the uniform consensus problem, and the formalization of the adaptive condition-based approach. In Section 3 and 4, we present our adaptive condition-based consensus algorithm. This algorithm is proposed in the incremental manner: In Section 3, we first introduce a simple adaptive condition-based algorithm ACC, which is one of existing algorithms and the basis of the algorithm we propose in this paper. Then, in Section 4, we consider an improvement of ACC. Finally, we conclude this paper in Section 5.

2 Preliminaries

2.1 Distributed System

We consider a synchronous distributed system with round-based synchrony. The distributed system consists of n processes $P = \{p_0, p_1, p_2, \cdots, p_{n-1}\}$ that are completely connected, that is, any pair of processes can communicate with each other by directly exchanging messages. All channels are reliable: each channel correctly transfers messages. The system is round-based, that is, its execution is a sequence of synchronized *rounds* identified by $1, 2, 3 \cdots$. Each round r consists of three phases:

Send phase. Each process p_i sends messages.
Receive phase. Each process p_i receives all the messages sent to p_i at the beginning of round r.
Local processing phase. Each process p_i executes local computation.

Processes can crash. If a process p_i crashes during round r, it makes no operation subsequently. The messages sent by p_i at round r may or may not be received. We say a process is *correct* if it never crashes, and say "a round r is correct" when no process crashes during round r. There are an upper bound t on the number of processes that can crash. We also denote the actual number of crash processes by f ($\leq t$). In the rest of the paper, we assume that $t < n/2$ holds.

2.2 Uniform Consensus

In a consensus algorithm, each correct process initially proposes a value, and eventually chooses a decision value from the values proposed by processes so that all processes decide the same value. The *uniform consensus* is a stronger variant of the consensus. It disallows faulty processes to disagree on the decision value. More precisely, the uniform consensus is specified as follows:

Termination **:** Every correct process eventually decides.
Uniform Agreement **:** No two processes decide different values.
Validity **:** If a process decides a value v, then, v is a value proposed by a process.

The set of values that can be proposed is denoted by $\mathcal{V}$. Moreover, we assume that $\mathcal{V}$ is a finite ordered set.

2.3 Legality Level

Notations. An *input vector* is a vector in $\mathcal{V}^n$, where the i-th entry represents p_i's proposal value. We usually denote an input vector for an execution by I. We also define *view* J of I to be a vector in $(\mathcal{V} \cup \{\perp\})^n$ obtained by replacing some entries in I by $\perp$ ($\perp$ is a default value such that $\perp \notin \mathcal{V}$). Let $\perp^n$ be the view such that all entries are $\perp$. For views J_1 and J_2, we denote $J_1 \leq J_2$ if $\forall k : J_1[k] \neq \perp \Rightarrow J_1[k] = J_2[k]$ holds. For two views J_1 and J_2, we define their union $J = J_1 \cup J_2$ as follows: $\forall k : J[k] = a \neq \perp \Leftrightarrow J_1[k] = a$ or $J_2[k] = a$. For a view J ($\in (\mathcal{V} \cup \{\perp\})^n$) and a value $a (\in \mathcal{V} \cup \{\perp\})$, $\#_a(J)$ denotes the number of entries of value a in the vector J, that is $\#_a(J) = |\{k \in \{0, 1, \cdots, n-1\} | J[k] = a\}|$. For a view J and a value a, we often describe $a \in J$ if there exists a value k such that $J[k] = a$. Finally, for two vectors J_1 and J_2, we denote the Hamming distance between J_1 and J_2 by $\mathrm{dist}(J_1, J_2)$, that is $\mathrm{dist}(J_1, J_2) = |\{k \in \{0, 1, \cdots, n-1\} | J_1[k] \neq J_2[k]\}|$.

Conditions and Legality. A *condition* is formally defined as a subset of $\mathcal{V}^n$. First, as an important class of conditions, we introduce (d, h)-legal conditions [2].

Definition 1 ((d, h)-legal conditions). *A condition C is (d, h)-legal (where h is a mapping $h : C \mapsto \mathcal{V}$) if h, d, and C satisfy the following properties:*

1. $\forall I \in C : \#_{h(I)}(I) > d$,
2. $\forall I_1, I_2 \in C : h(I_1) \neq h(I_2) \Rightarrow \mathrm{dist}(I_1, I_2) > d$.

Intuitively, (d, h)-legal condition is the set of input vectors I such that $h(I)$ can be calculated even when at most d entries of I are lost. From the definition, as long as mapping h satisfies $h(I) \in I$, $\mathcal{V}^n$ can be $(0, h)$-legal, and (n, h)-legal condition is only the empty set. Notice that (d, h)-legal condition is not uniquely determined by d and h (for instance, for a (d, h)-legal condition, its subset is also a (d, h)-legal condition). In recent researches, it is shown that (d, h)-legal conditions help to reduce the worst-case execution time of synchronous consensus algorithms. To be more precise, for any (d, h)-legal condition, there exists a consensus algorithm that terminates (1) within $\min\{t + 1 - d, f + 2\}$ rounds for

[2] The (d, h)-legal conditions is a subclass of d-legal conditions defined in [15] (the condition C is d-legal if there exists a mapping h such that C is (d, h)-legal). This difference does not restrict the class of condition applicable to our algorithm because our algorithm can be instantiated with any h.

input vectors satisfying the condition, and (2) within $\min\{f+2, t+1\}$ rounds otherwise [16]. In this sense, we can regard d as a characteristic value representing difficulties of input vectors in (d,h)-legal condition. However, from the definition, a (d,h)-legal condition can include a $(d+1,h)$-legal condition. This implies that a (d,h)-legal condition can include easier input vectors. Therefore, to define actual difficulty of input vectors, we introduce *legality levels* of input vectors that are defined from a *legal condition sequence* as follows:

Definition 2 (Legal condition sequence). *A sequence of conditions $\mathcal{C} =< C_0, C_1, \cdots C_n >$ is an h-legal condition sequence if the following properties are satisfied:*

- $C_0 = \mathcal{V}^n$, $C_n = \emptyset$,
- $\forall k$ *(*$0 \leq k \leq n-1$*):* C_k *is* (k,h)*-legal and* $C_{k+1} \subseteq C_k$*, and*
- $\forall k$ *(*$0 \leq k \leq n-1$*):* $(\nexists C' : C'$ *is* $(k+1,h)$*-legal and* $C_{k+1} \subset C' \subseteq C_k)$*.*

Definition 3 (Legality level). *For a h-legal condition sequence $\mathcal{C}$, the legality level of an input vector I is l if $I \in C_l$ and $I \notin C_{l+1}$ hold.*

Since C_n is empty and C_0 is the set of all possible input vectors, any legal condition sequence uniquely defines the legality level of each input vector. The legality level represents the actual difficulties of input vectors in the sense that we previously mentioned.

Example An example of a (d,h)-legal condition is $C_d^{\max}$:

$$C_d^{\max} = \{I \in \mathcal{V}^n | \#_a(I) > d, \text{ where } a \text{ is the maximum value in } I\}$$

The condition $C_d^{\max}$ is a maximal (d,h)-legal condition defined by d and $h = \max$. Moreover, it is *maximal*, that is, there is no (d,h)-legal condition C such that $C_d^{\max} \subset C$ [14] [3]. Therefore, for $C_d^{\max}$, we can define legal condition sequence $\mathcal{C}^{\max} =< C_0^{\max}, C_1^{\max}, \cdots C_n^{\max} >$. As an example, we consider two input vectors, $I_1 =< 0,0,1,3,3 >$ and $I_2 =< 0,0,2,2,2 >$. Both vectors are contained in $C_1^{\max}$. However, whereas I_2 is contained in $C_2^{\max}$, I_1 is not. Therefore, for $\mathcal{C}^{\max}$, legality levels of vectors I_1 and I_2 are respectively 1 and 2.

The algorithm proposed in this paper is instantiated with a legal condition sequence, that is, the instantiated algorithm can utilize the legal condition sequence and the mapping h. In the following discussions, let the algorithm be instantiated with a legal condition sequence $\mathcal{C} =< C_0, C_1, \cdots, C_n >$, where each C_k is (k,h)-legal. We denote the legality level of an input vector I for $\mathcal{C}$ by $l(I)$.

[3] Actually, the definition of maximality in [14] is stronger: The (d,h)-legal condition C is maximal if $C \cup \{I'\}$ is not (d,h')-legal for any mapping h' and input vector $I' \notin C$

3 Algorithm ACC

The algorithm we present in this paper is an extension of algorithm ACC presented in [12]. Thus, in this section, we first introduce the algorithm ACC. The algorithm ACC terminates within $f + 2 - l(I)$ rounds if $l(I) < f$, within two rounds if $l(I) \geq f$ and $f \neq 0$ holds, and in one round if $l(I) \geq t$ and $f = 0$ holds [4]. In the following subsection, we first introduce the fundamental function decode, which is used as a subroutine of our algorithm. Then, we propose the algorithm ACC.

3.1 Function decode

The function decode(J) is presented in Figure 1. The function decode has an argument J which is a view. Informally, the role of the function decode(J) is to obtain the value $h(I)$ from J. When a process invokes decode(J), it constructs all possible vectors I' such that $I' \geq J$ holds (line 4), and chooses the vector with the maximum legality level from them (line 5). For the chosen vector I', the algorithm returns the value $h(I')$. For this function, we can show that the following lemmas hold. In the following lemmas, let $\mathcal{E}(J)$ be the value stored in $\mathcal{E}$ immediately after the line 4 of decode(J) is executed.

Lemma 1. *If $\mathcal{E}(J)$ is nonempty, the condition* $\mathrm{dist}(I_1, I_2) \leq \#_{\perp}(J)$ *holds for any I_1, $I_2 \in \mathcal{E}(J)$.*

Proof. *Since $J \leq I_1$ and $J \leq I_2$, this lemma clearly holds.* □

Lemma 2 (Decode Validity). *For any $J \neq \perp^n$, the resultant value of* decode(J) *is contained in J.*

Proof. *For any vetror $I \in \mathcal{E}(J)$, $h(I) \in J$ holds. If $\mathcal{E}(J) \neq \emptyset$, the resultant value $h(I')$ is necessarily contained in J because the vector I' belongs to $\mathcal{E}$. Thus, we prove this lemma by showing $\mathcal{E}(J) \neq \emptyset$ for any $J \neq \perp^n$. Suppose for contradiction that for a view J, the set $\mathcal{E}(J)$ becomes empty. Then, we consider a vector $I'' \geq J$ that is obtained by replacing $\perp$ of J by a value in J. From the definition of mapping h, $h(I'')$ is a value in I'', that is, it is a value in J. This implies that $I'' \in \mathcal{E}(J)$ holds. This is contradiction.* □

Proof. *Clearly, return values are included in J. We prove the termination by showing that* decode(J) *is necessarily terminates. Since we assume that $J \neq \perp^n$, $\mathcal{E}(J)$ is necessarily nonempty (any vector obtained by replacing $\perp$ by a non-$\perp$ value in J is necessarily contained in $\mathcal{E}(J)$), and thus the execution terminates.* □

[4] The algorithm ACC presented here is slightly different from that in [12]. The original ACC does not have one-round decision scheme (however, this scheme itself is also presented in [12]).

```
1: Function decode(J) :
2:   variable
3:     E : init ∅

4:   E ← {I ∈ V^n | h(I) ∈ J and J ≤ I}
5:   I' ← argmax_{I'∈E} l(I')
6:   return(h(I'))
```

Fig. 1. Function decode(J)

Lemma 3 (Decode Agreement). *Let I be an input vector, and J be a view such that $J \leq I$ and $\#_\perp(J) \leq l(I)$ hold. Then,* decode$(J) = h(I)$ *holds.*

Proof. *Let I' be the vector that is chosen from $\mathcal{E}$ at line 5 (that is, the resultant value is $h(I')$). Then the vector I and I' are necessarily included in $\mathcal{E}$, and thus* $\mathrm{dist}(I', I) \leq \#_\perp(J) \leq l(I)$ *holds from lemma 1. In addition, since the vector I' has the maximum legality level in $\mathcal{E}$, $l(I) \leq l(I')$ also holds. This implies that $I' \in C_{l(I)}$ holds. Therefore, $h(I') = h(I) =$*decode(J) *holds.* □

3.2 The Behavior of Algorithm ACC

Using function decode, we proposed a simple adaptive algorithm ACC [12]. The algorithm is based on the well-known floodset algorithm [13][18]. The typical floodset algorithm is as follows: Each process maintains its own view, which stores only its proposal at round 1. In each round, each process sends its own view to all processes, receives views from other processes, and updates its own view by the union of the current view and all the received views. The primary objective of the floodset algorithm is to guarantee that all processes have a same view after execution of an appropriate number of rounds. In non-condition-based algorithms, $f + 1$ rounds are sufficient for all non-crashed processes to have a same view. This relies on the fact that $f + 1$ rounds' execution includes at least one correct round and the fact that all non-crashed processes have a same view at the end of a correct round. On the other hand, considering the input vector condition, $f + 1 - l(I)$ rounds are sufficient [16][19]. In this case, processes may have different views at the end of round $f+1-l(I)$, however, it is guaranteed that a common value (that is $h(I)$) can be calculated from each view. The primary issue the algorithm ACC must consider is to execute the floodset algorithm till an appropriate round even though the values of f and $l(I)$ are unknown.

The behavior of ACC is as follows: The algorithm executes the floodset algorithm as an underlying task. In each round r, each process p_i supposes that legality level of the input vector is $t + 1 - r$, and estimates a decision value by executing decode(J_i), where J_i is the view maintained by p_i. This estimation can be wrong, and thus, at the next round, each process checks whether its estimation is correct or not. More precisely, at round $r + 1$, each process p_i sends its estimation to all processes (including itself). If all messages received by p_i have a same estimation w, p_i decides a value w. Then, each process terminates at round $f + 2 - l(I)$ or earlier. However, if a process p_j ac-

cidentally decides at a round earlier than round $f + 2 - l(I)$ while another process p_i decides at round $f + 2 - l(I)$, those decisions may differ. Hence, to avoid this inconsistency, we introduce the scheme of overwriting views into the algorithm: If p_i receives more than $n/2$ messages carrying a common estimation w, before the estimation for the next round, it overwrites its own view by the view J from other processes such that $\mathsf{decode}(J) = w$ holds. When a process decides a value w at round $r + 1$, all other processes necessarily have such a view as J at the end of round $r + 1$ because at least $n - f(> n/2)$ correct processes necessarily sends the same estimation w at round $r + 1$. Thus, all other processes are guaranteed to decide the value w at round $r + 2$.

In addition, the algorithm ACC has another exceptional decision scheme that allows the algorithm to terminate in one round if $l(I) \geq t$ and $f = 0$ hold. We call this scheme *fast decision*. The idea of the fast decision is borrowed from [15]. At round 1, if a process p_i gathers all proposals and recognizes that legality level of the input vector is t or more, it does not have to execute the next round for checking its estimation. Therefore, the process p_i can immediately decide a value $\mathsf{decode}(J_i)$ $(= h(I))$ and thus, the algorithm terminates if $f = 0$. In this case, even though up to t processes crash, all other processes can calculate $h(I)$ from their own views. This implies that each process eventually decides a value $h(I)$, and thus the uniform agreement is guaranteed.

Figure 2 presents the code of the algorithm ACC for process p_i. The view of each process p_i is maintained in the variable J_i. The variables $Views_i$ and S_i respectively denote views and estimations received from other processes at the current round. The lines 9-11 correspond to the fast decision scheme. The lines 12-18 correspond to the view-overwriting scheme. The line 19 corresponds to the estimation of a decision value. Notice that the estimation is done after the view-overwriting.

4 Improved Algorithm IACC

In this subsection, we present the algorithm IACC, which is an extension of ACC. The algorithm IACC terminates within $\min\{f + 2, t + 1\} - l(I)$ rounds when $l(I) < t$ holds. Compared with ACC, it improves the time complexity by one round in the case of $f = t$ and $l(I) < f$.

4.1 Slow Decision

To improve the time complexity, the algorithm IACC has one more additional decision scheme, which is called *slow decision*. The algorithm IACC is presented in Figure 3, which is obtained by appending extra codes of the slow decision scheme to ACC. The slow decision part appear at lines 23-27. In addition, messages are modified to carry an extra information num. Since the algorithm ACC clearly

```
Algorithm ACC(v_i) for h-legal condition sequence and t crashes (t < n/2)
Code for p_i:

1:  variable:
2:      J_i, S_i : init ⊥^n and  J_i[i] ← v_i
3:      s_i : init ⊥
4:      Views_i : init <⊥^n, ⊥^n, ··· , ⊥^n>

5:  for each round r = 1, 2, ··· , t + 2 do :
6:      send (J_i, s_i) to all processes (including p_i)
7:      Let (Views_i[j], S_i[j]) be the message received from p_j
        (if no message is received from p_j, Views_i[j] =⊥^n and S_i[j] =⊥)
8:      J_i ← ⋃_{k=0}^{n-1} Views_i[k]                          /* Updating the view */
9:      if r = 1 and #_⊥(J_i) = 0 and l(J_i) ≥ t then
10:         decide(decode(J_i)) and exit                        /* Fast decision */
11:     endif
12:     if r > 1 then
13:         if ∃w ≠⊥: #_w(S_i) + #_⊥(S_i) = n then decide(w) and exit endif
14:         if ∃w ≠⊥: #_w(S_i) > n/2 then
15:             Let y be a value such that S_i[y] = w and #_w(S_i) > n/2 hold
16:             (deterministically chosen)
17:             J_i ← Views_i[y]                                /* Overwriting the view */
18:         endif
19:     endif
20:     s_i ← decode(J_i) /* Estimation of decision value */
21: endfor
```

Fig. 2. Algorithm ACC: Adaptive Condition-based Consensus

achieves the improved time complexity in the case of $f < t$ or $l(I) \geq f$, at the following discussion, we assume that $f = t$ and $l(I) < f$ holds.

The basic idea of the slow decision is derived from a precedent result [16]. Our scheme can be regarded as an adaptive version of it.

The aim of the slow decision scheme is to stop execution at round $t + 2 - f'$, where f' is the number of crash faults occurring at round one. It can be understood that this idea achieves $t + 1 - l(I)$-round time complexity although the consensus problem is correctly solved by the following observation: If f' is $l(I)$ or less, each process can calcutate $h(I)$ at round one and thus they can reach agreement within two rounds. On the other hand, if f' is greater than $l(I)$ then, $t + 2 - f'$-round execution contains at least one correct round, and thus all processes can reach agreement at round $t + 2 - f'$ or earlier. In both cases, most importantly, the length of the execution does not exceed $t+1-l(I)$ rounds (remind that we only consider the case of $l(I) < t$).

To achieve this aim, in our scheme, each process detects the number of crash faults occurring at round one, and broadcasts the number of detected crashes as an extra information num. Each process p_i has the variable num_i, which represents the estimation of f'. At round two or later , each process stores the maximum value of all received num into num_i as the current estimation of f'. When current round r is larger than or equal to $t+2-num_i$, process p_i decides its current estimation s_i and terminates. Then, each estimation num_i can be slightly different from actual value of f'. However, this difference does not cause any problem. The details are explained in the correctness proof (Case2-2 and Case2-3 in the proof of Lemma 8).

Algorithm ACC(v_i) for h-legal condition sequence and t crashes ($t < n/2$)
Code for p_i:

```
1:  variable:
2:      J_i, S_i : init ⊥^n and  J_i[i] ← v_i
3:      s_i : init ⊥
4:      num_i : init 0
5:      Views_i : init <⊥^n, ⊥^n, ···, ⊥^n>
6:      F_i : init < 0, 0, ···, 0 >

7:  for each round r = 1, 2, ···, t + 2 do :
8:      send (J_i, s_i, num_i) to all processes (including p_i)
9:      Let (Views_i[j], S_i[j], F_i[j]) be the message received from p_j
        (if no message is received from p_j, Views_i[j] =⊥^n, S_i[j] =⊥, and F_i[j] = 0)
10:     J_i ← ⋃_{k=0}^{n-1} Views_i[k]                          /* Updating the view */
11:     if r = 1 and #_⊥(J_i) = 0 and l(J_i) ≥ t then
12:         decide(decode(J_i)) and exit                        /* Fast decision */
13:     endif
14:     if r > 1 then
15:         if ∃w ≠⊥: #_w(S_i) + #_⊥(S_i) = n then decide(w) and exit endif
16:         if ∃w ≠⊥: #_w(S_i) > n/2 then
17:             Let y be a value such that S_i[y] = w and #_w(S_i) > n/2 hold
18:             (deterministically chosen)
19:             J_i ← Views_i[y]                                /* Overwriting the view */
20:         endif
21:     endif
22:     s_i ← decode(J_i)                                       /* Estimation of decision value */
23:     if r = 1 then num_i ← #_⊥(J_i)
24:     else num_i ← max(F_i) endif                             /* Updating # faults of round 1 */
25:     if r ≥ t + 2 − num_i then
26:         decide(s_i) and exit                                /* Slow decision */
27:     endif
28: endfor
```

Fig. 3. Algorithm IACC: An Improved Adaptive Condition-based Consensus Algorithm

4.2 Correctness of IACC

In this subsection, we prove the correctness of IACC. For the proof, we define the following notations and terms: J_i^r, $Views_i^r$, num_i^r and S_i^r respectively denote the value of J_i, $Views_i$, num_i^r and S_i at the end of round r (line 27). Let P^r be the set of processes that neither crash by the end of round r nor terminate by the end of round $r-1$ (possibly terminate at the end of round r), and P_c be the set of correct processes. For short, let $f_m = \max\{num_i | p_i \in P^1\}$. We say that p_i decides by the *fast decision*, the *slow decision* or the *regular decision* if p_i decides at line 12, 15, or 26 respectively. Notice that the regular decision has higher priority than the slow decision. That is, if a process can decide by both the regular and the slow decision, it decides by the regular decision. Due to lack of space, proofs of several lemmas are omitted. They are given in Appendix.

Lemma 4 (Validity). *If a process decides a value w, then w is a value proposed by a process.*

Proof. *This lemma clearly holds from Lemma 2.* □

Lemma 5. *If a round r $(1 \leq r \leq t+1)$ is correct, then $J_i^k = J_j^k$ holds for any $p_i, p_j \in P^k$ and $k \geq r$.*

Proof. *We prove this lemma by induction on k. (***Basis***) We consider the case of $k = r$. Since round $k(= r)$ is correct, each process in P^k receives a same set of messages at round k. Thus, for any $p_i, p_j \in P^k$, $Views_i^k = Views_j^k$ and $S_i^k = S_j^k$ holds. Since the value of J_i^k is deterministically calculated from the values of $Views_i^k$ and S_i^k, $J_i^k = J_j^k$ holds for any $p_i, p_j \in P^k$. (***Inductive step***) Suppose as induction hypothesis that $J_i^k = J_j^k$ holds for some $k(\geq r)$ and any $p_i, p_j \in P^k$ (let J be the value of J_*^k). Since each process in P^k sends a message $(J, *)$ at round $k+1$ unless it crashes, for each p_i, $Views_i^{k+1}$ contains only values J and $\perp^n$. Then, the value of J_i^{k+1} is either $\bigcup_{x=0}^{n-1} Views_i^{k+1}[x] = J$ (obtained at line 10) or $J^{k+1} = Views_i^{k+1}[x] = J$ (obtained at line 18). In any cases, $J_i^{k+1} = J$ holds. This implies that $J_i^{k+1} = J_j^{k+1}$ holds for any $p_i, p_j \in P^{k+1}$.* □

Lemma 6. *If a round r $(1 \leq r \leq t+2)$ is correct, then every process $p_i \in P^{r+1}$ decides at round $r+1$ by the regular decision.*

Proof. *From Lemma 5, the variable J_i^r has a common view (say J) for any $p_i \in P^r$. This implies that each process p_i sends the same message (J, w, num_i^{r-2}) at round $r+1$ where $w = \mathsf{decode}(J)$. Then, since S_i^{r+1} contains only w and $\perp$, each process $p_i (\in P^{r+1})$ decides a value w at round $r+1$ by the regular decision.* □.

Lemma 7 (Regular Termination). *Each process p_i decides a value at round $\max\{2, f+2-l(I)\}$ or earlier unless it crashes.*

Proof. *Let $R = \max\{2, f+2-l(I)\}$. If there exists a correct round r up to $R-1$, the lemma clearly holds from Lemma 6. Thus, we have only to consider the case that every round up to $R-1 (\leq \max\{1, f+1-l(I)\})$ is not correct. Since at least one process crashes in each round up to $R-1$, at most $l(I)$ processes can crash at round 1 (notice that it also holds in the case of $1 \geq f+1-l(I)$). Then, $\#_\perp(J_i^{R-1}) \leq l(I)$ holds for any $p_i \in P^{R-1}$ because if a process p_k does not crash at round 1, all processes receive p_k's proposal at round 1, and thus, $J[k] \neq \perp$ holds for every view in the execution. Therefore, from Lemma 3, we obtain $\mathsf{decode}(J_i^{R-1}) = h(I)$. Then, since every process p_i in P^{R-1} sends message $(J_i^{R-1}, h(I))$, S_i^R contains only $h(I)$ and $\perp$. This implies that each process p_i in P^R decides $h(I)$ at round R.* □

Lemma 8 (Slow Termination). *If $l(I) < t$ holds, each process p_i decides a value at round $t+1-l(I)$ or earlier unless it crashes.*

Proof. *We consider the following two cases.*

- *(***Case1***) When $f_m \leq l(I)$ holds: Since $\perp$ appears at most f_m times in J_k^1, $\#_\perp(J_k^1) \leq f_m \leq l(I)$ holds for any process $p_k \in P^1$. Then, from Lemma 3, we obtain $\mathsf{decode}(J_k^1) = h(I)$, and thus each process $p_k \in P^1$ sends a message $(View_k^1, h(I), num_k^1)$ at round 2. This implies that S_k^2 contains only values $h(I)$ and $\perp$ for any $p_k \in P^2$. Therefore, each process decides at round 2 or earlier, that is, each process decides at round $t+1-l(I)$ or earlier (because we assume $t > l(I)$).*
- *(***Case2***) When $f_m > l(I)$ holds: Since at least f_m processes crash at round 1, there exists a correct round in the first $t+2-f_m$ rounds. Letting r' be such a correct round, for any p_i, each $num_i^{r'}$ has a common value f'_m because each p_i receives a same set of messages at round r' and the value of $num_i^{r'}$ is the maximum vale among all the received values of num_*. In addition, this also implies that $num_i^{r''} = f'_m$ holds for any $r'' \geq r'$. It follows that $num_k^{t+1-l(I)} = f'_m$ for any $p_k \in P^{t+1-l(I)}$ (remind that $t+2-f_m \leq t+1-l(I)$ holds because we assume $f_m > l(I)$). Here, we further divide this case into the following three cases (notice that $f_m \geq f'_m$ holds).*
  - *(***Case2-1***) When $f_m \geq f'_m > l(I)$ holds: In this case, at round $r = t+1-l(I)$, $r \geq t+2-f'_m = t+2-num_i^r$ holds for any $p_i \in P^r$. This implies that each process decides at round $t+1-l(I)$ or earlier (line 25).*
  - *(***Case2-2***) When $f_m > l(I) \geq f'_m$ and $t+1-l(I) \geq 3$ hold: Then, $num_i^1 \leq f'_m$ holds for any $p_i \in P_c$ (if not, f'_m becomes larger because a process in P_c sends a message with higher num_* at round two and all processes in P^2 receive it). Hence, from Lemma 3, $\mathsf{decode}(J_i^1) = h(I)$ holds for any process $p_i \in P_c$. Since we assume $l(I) < t$, each process in P_c does not decide at round 1 and sends a message containing $h(I)$ to all processes at round 2. Then, $\#_{h(I)}(S_i^2) \geq |P_c| > n/2$ holds for any $p_i \in P^2$. This implies that each process in P^2 overwrites its view by the view J such that $\mathsf{decode}(J) = h(I)$, and that sends a message containing $h(I)$. Therefore, S_i^3 contains only $h(I)$ and $\perp$ for any $p_i \in P^3$, and thus we can conclude each p_i decides at round $3 \leq t+1-l(I)$.*
  - *(***Case2-3***) When $f_m > l(I) \geq f'_m$ and $t+1-l(I) < 3$ hold: Then, since $t-2 < l(I) < f_m$ holds, we obtain $f_m = t$ and $l(I) = t-1$. This implies that t processes crash at round 1 and thus round 2 is correct. Therefore, at the end of round 2, $num_i^2 = f_m = t$ holds for any $p_i \in P^2$, and each process decides at round $2 = t+1-l(I)$.*

□

Lemma 9 (Fast Termination). *If $l(I) \geq t$ and $f = 0$ holds, each process p_i decides a common value at round 1.*

Proof. *Since $f = 0$ holds, each process p_i receives messages from all processes. This implies that $J_i^1 = I$ holds, and thus, p_i decides $\mathsf{decode}(I)$ at round 1 (line 12).* □

Lemma 10. *Let p_i be the process that decides first. If p_i decides w at round r by the regular decision, then each process $p_k \in P_c$ sends a message $(J_k^{r-1}, w, num_k^{r-1})$ at round r.*

Proof. *Since the process p_i decides a value w by the regular decision at round r, S_i^r contains only w and $\perp$. This implies that every process $p_k \in P_c$ sends a message $(J_k^{r-1}, w, num_k^{r-1})$ at round r because no process in P_c terminates at the beginning of round r (remind that p_i is the process that decides first).* □

Lemma 11. *Let p_i be the process that decides first. If p_i decides w at round r, then each process $p_j \in P^r$ decides w at round r or $r+1$ unless it crashes.*

Proof. *We consider the following two cases. (***Case1***) When p_j decides at round r: From Lemma 10, clearly $S_j^r[k] = w$ holds for any process $p_k \in P_c$, and thus p_j decides w. (***Case2***) When p_j does not decide at round r: From Lemma 10, we can show that every process in P_c sends a message $(J_*^{r-1}, w, num_*^{r-1})$ at round r. Then, since $t < n/2$ holds, we obtain $\#_w(S_k^r) > n/2$ for any $p_k \in P^r$. Therefore, each process p_k overwrites its own variable J_k by a view V_k such that $\mathsf{decode}(V_k) = w$ holds. Then, since every process $p_k \in P^r$ sends a message (V_k, w, num_k^{r+1}) at round $r+1$ (or is crashed at the beginning of round $r+1$), S_j^{r+1} contains only w and $\perp$ for any $p_j \in P^{r+1}$. This implies that p_j decides a value w at round $r+1$.* □.

Lemma 12 (Uniform Agreement). *No two processes decide different values.*

Proof. *Let p_i be the first process that decides, and w be p_i's decision. We show that any process p_j decides the same value w if it decides. Let r denote the round when p_i decides.*

- *(***Case1***) When p_i decides by the fast decision: Since p_i decides at round one, $l(I) \geq t$ and $w = h(I)$ clearly holds. Then, from Lemma 3, $\mathsf{decode}(J_k^1) = h(I) = w$ holds for each $p_k \in P^1$ since $\#_\perp(J_k^1) \leq t \leq l(I)$ holds. Thus, if p_j decides by the fast decision, it clearly decides w. Even if not, p_j also decides w at round two unless it crashes because S_j^2 contains only w and $\perp$ at the end of round two.*
- *(***Case2***) When both p_i and p_j decide by the regular decision : In this case, this lemma clearly holds from Lemma 11.*
- *(***Case3***) When p_i and p_j decide respectively by the regular decision and the slow decision : From Lemma 11, p_j decides at round $r+1$ by the regular decision if it does not decide at round r. Since we assume p_j decides by the slow decision, p_j decides at round r. On the other hand, from Lemma 10, we can show that every process $p_k \in P_c$ sends a message $(J_k^{r-1}, w, num_k^{r-1})$ at round r, and thus $\#_w(S_j^r) > n/2$ holds. Then, before the decision, p_j overwrites its own variable J_j by a view V_k such that $\mathsf{decode}(V_k) = w$ holds, and assigns w to s_j. This implies that p_j also decides w.*

– *(**Case4**) When p_i decides by the slow decision : The proof of this case consists of three steps. We first show that the round r is a correct round. Suppose for contradiction that r is not correct. Since at least num_i^r processes crash at round 1, there exists a correct round r' by round $t + 2 - num_i^r (\leq r)$. Since we assume that r is not correct, $r' < r$ holds. Then, from Lemma 6, p_i decides at round $r' + 1 (\leq r)$ by the regular decision. This is contradiction.*

Next we prove that p_j never decides at round r by the regular decision. Suppose for contradiction that p_j decides at round r by the regular decision. Then, since round r is correct, p_i also receives the same set of messages as p_j. Thus, $S_j^r = S_i^r$ holds. This implies that p_i also decides at round r by the regular decision. This is contradiction.

Finally, we prove that p_j decides w at round r by the slow decision. Since round r is correct, each process $p_k \in P^r$ receives a same set of messages at round r. This implies that $num_j^r = num_i^r$ holds. Thus, in this case, p_j also decides at round r by the slow decision because the second step above shows p_j does not decide at round r by the regular decision. Moreover, we obtain $J_i^r = J_j^r$ from Lemma 5. This implies that $\mathsf{decode}(J_j^r) = decode(J_i^r) = w$ holds and thus p_j decides w.

From the above, the lemma holds. □

From Lemmas 4, 7, 8, 9 and 12, the following theorem holds.

Theorem 1. *The algorithm* IACC *solves the uniform consensus (1) within one round if $l(I) \geq t$ and no process crashes, (2) within two rounds if $l(I) \geq f$, and (3) within $\min\{f + 2, t + 1\} - l(I)$ rounds otherwise.*

5 Concluding Remarks

This paper proposed an adaptive condition-based algorithm for synchronous consensus. The algorithm improved the time complexity of our previous results in [12]. On the assumption that majority of processes is correct, the proposed algorithm terminates within $\min\{f + 2, t + 1\} - l(I)$ rounds if $l(I) \leq f$, and within 2 rounds if $l(I) \geq f$, where $l(I)$ is the legality level of the input vector. Moreover, this algorithm terminates with one round if $l(I) \geq t$ and $f = 0$ hold (fast decision), Compared with existing algorithms, it achieves the best time complexity.

Acknowledgment. This work is supported in part by a JSPS, Grant-in-Aid for Scientific Research ((B)(2)15300017), and "The 21st Century Center of Excellence Program" of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. H. Attiya and J. L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.
2. M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proc. of the second annual ACM symposium on Principles of distributed computing(PODC)*, pages 27–30, 1983.
3. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
4. B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. *Journal of Algorithms*, 51:15–37, Apr 2004.
5. D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.
6. D. Dolev, R. Reischuk, and R. Strong. Early stopping in byzantine agreement. *Journal of ACM*, 37(4):720–741, 1990.
7. C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.
8. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
9. R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and consensus. In *Proc. of 9th International Workshop on Distributed Algorithms(WDAG)*, volume 972 of *LNCS*, Sep 1995.
10. V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.
11. M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13:124–149, 1991.
12. T. Izumi and T. Masuzawa. Synchronous condition-based consensus adapting to input-vector legality. In *Proc. of 18th International Conference on Distributed Computing(DISC)*, volume 3274 of *LNCS*, pages 16–29. Springer-Verlag, Oct 2004.
13. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
14. A. Mostefaoui, S. Rajsbaum, and M. Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922–954, 2003.
15. A. Mostefaoui, S. Rajsbaum, and M. Raynal. Using conditions to exppedite consensus in synchronous distributed systems. In *Proc. of 17th International Conference on Ditributed Computing(DISC)*, volume 2848 of *LNCS*, pages 249–263, Oct 2003.
16. A. Mostefaoui, S. Rajsbaum, and M. Raynal. The synchronous condition-based consensus hierarchy. In *Proc. of 18th International Conference on Distributed Computing(DISC)*, volume 3274 of *LNCS*, pages 1–15. Springer-Verlag, Oct 2004.
17. A. Mostefaoui, S. Rajsbaum, M. Raynal, and M. Roy. Condition-based consensus solvability: a hierarchy of conditions and efficient protocols. *Distributed Computing*, 17(1):1–20, 2004.
18. M. Raynal. Consensus in synchronous systems: A concise guided tour. In *Proc. of Pacific Rim International Symposium on Dependable Computing(PRDC)*, pages 221–228, 2002.
19. Y. Zibin. Condition-based consensus in synchronous systems. In *Proc. of 17th International Conference on Ditributed Computing(DISC)*, volume 2848 of *LNCS*, pages 239–248, Oct 2003.

Biangular Circle Formation by Asynchronous Mobile Robots*

Branislav Katreniak

Faculty of Mathematics, Physics and Informatics,
Comenius University, Bratislava
katreniak@dcs.fmph.uniba.sk

Abstract. Consider a community of simple autonomous robots freely moving in the plane. The robots are decentralized, asynchronous, deterministic without the common coordination system, identities, direct communication, memory of the past, but with the ability to sense the positions of the other robots. We study the problem of forming an absolutely symmetric formation – regular circle. Unlike the existing algorithms for similar problems that have supposed a stronger model and guaranteed only convergence to a final formation, we are interested in solving the problem for fully asynchronous model. Unfortunately, the problem in general is very hard under these circumstances and seems to be unsolvable. We present an algorithm that solves an intermediate problem, the biangular circle formation, deterministically in finite time.

1 Introduction

We consider a distributed system consisting of very weak autonomous mobile robots. The robots are anonymous, have no common knowledge, no common sense of the direction (e.g. compass), no central coordination and no means of direct communication. The study of such a weak system is motivated by the question of the minimal complexity of the mobile devices needed to perform non-trivial tasks. While a number of results deal with heuristics and practical applications, deterministic worst-case solutions are rare. We use the asynchronous model introduced in [4] where the task of arbitrary pattern formation is addressed. It was shown that with a common sense of the direction the arbitrary pattern formation problem is solvable. Moreover, without the common sense of the direction there are patterns which are not formable. In [2], [3] it was shown that the gathering problem is solvable with multiplicity detection and in limited visibility model with compass. We use the basic model without any extensions and study the problem of the pattern formation for the particular pattern – biangular circle. The only similar problem (convergence to the regular circle pattern) was addressed [1] only in a semi-synchronous model.

* Supported in part by APVT 20-018902.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 185–199, 2005.

Consider the life-cycle of a robot: Initially it is in a waiting state, wakes up asynchronously, observes the other robots' positions, computes a point in the plane, moves toward this point (but may not reach it) and becomes waiting again. Each step takes an unpredictable amount of time. Robots are oblivious, i.e. the only input for their computation is the currently observed set of the other robots' positions.

First, we only require the robots to move onto the boundary of a *circle*. This problem is easy, every robot just takes the shortest way to the smallest enclosing circle (SEC) of all robots. The SEC will never move and all robots will reach SEC in a finite time.

In section 4 we show how the robots can reach the boundary of a circle at *distinct positions*, provided they are located at distinct positions at the start. Only the robots closest to SEC will take the shortest way to SEC, others will wait until it is secure for them to do a *side-step* – an atomic move along the concentric circle.

Finally, we would like the robots to form[1] a *regular circle*. This problem is very hard and seems to be unsolvable in general. In section 5 we show an intermediate result: Robots try to form a regular circle, but they do not achieve it in all cases. In general they form only a bit less symmetric pattern, the *biangular circle*. The main idea of the algorithm is the synchronization of the asynchronous robots. We present a restricted pseudo-synchronous model with robots moving along the boundary of a fixed circle and show its emulation in our asynchronous model. Then this pseudo-synchronous circle model is used to transform the circle pattern into the biangular circle pattern.

2 Model

We use the model introduced in [4]. Each robot is viewed as a point in a plane equipped with sensors. It can observe the set of all points which are occupied by at least one other robot. Note that the robot only knows whether there are other robots at a specific point, but it has no knowledge about their number (i.e. it cannot tell how many robots are at a given location). The *local view* of each robot consists of a unit of length, an origin (w.l.o.g. the position of the robot in its current observation), an orientation of angles and the coordinates of the observed points. No kind of agreement on the unit of length, the origin or the orientation of angles is assumed among the robots.

A robot is initially in a *waiting* state (*Wait*). Asynchronously and independently from the other robots, it *observes* the environment (*Look*) by activating its sensors. The sensors return a snapshot of the world, i.e. the set of all points occupied by at least one other robot, with respect to the local coordinate system. Then, based only on its local view of the world, the robot *calculates* its destination point (*Compute*) according to its deterministic algorithm (the same

[1] We again suppose that robots are at the start located at distinct positions.

for all robots). After the computation the robot *moves* towards its destination point (*Move*); if the destination point is the current location, the robot stays on its place. A move may stop before the robot reaches its destination (e.g. because of limits to the robot's motion energy). The robot then returns to the waiting state. The sequence *Wait* – *Look* – *Compute* – *Move* forms a *cycle* of a robot.

The robots are *fully asynchronous*, the amount of time spent in each phase of a cycle is finite but otherwise unpredictable. In particular, the robots do not have a common notion of time. As a result, robots can be seen by the other robots while moving, and thus computations can be made based on obsolete observations. The robots are *oblivious*, meaning that they do not remember any previous observations nor computations performed in any previous cycles. The robots are *anonymous*, meaning that they are indistinguishable by their appearance, and they do not have any kind of identifiers that can be used during the computation. Finally, the robots have *no means of direct communication*: any communication occurs in a totally implicit manner, by observing the other robots' positions.

There are two limiting assumptions concerning *infinity*:

Assumption 1. *The amount of time required by a robot to complete a cycle is finite and bounded from above be a fixed constant.*

Assumption 2. *The distance traveled by a robot in a cycle is bounded from below be a fixed ε (or the robot gets to the destination point).*

As no other assumptions on space exist, the distance traveled by a robot in a cycle is unpredictable.

3 Notations

In general, r denotes the robot itself or its position in the plane. The *configuration* of robots R at a given time is the set of positions in the plane occupied by the robots; n is the number of robots in R.

Given two distinct points a and b in the plane, $[a, b)$ denotes the half-line that starts in a and passes through b; $\texttt{flip}([a, b))$ denotes $[a, b)$ rotated by 180° about a; $[a, b]$ denotes the line segment between a and b; $|a, b|$ the Euclidean distance between the points a and b. Given two half-lines $[c, a)$ and $[c, b)$, we denote by $\sphericalangle(a, c, b)$ the clockwise angle centered in c from $[c, a)$ to $[c, b)$; by $|\sphericalangle(a, c, b)|$ the size of this angle; by $\texttt{axis}(\sphericalangle(a, c, b))$ the axis of this angle.

Given a circle C with center c and radius rad, we say that robot r *is on* C ($r \in C$) if $|r, c| = rad$ (i.e. r is on the circumference of C); r *is inside* C if $|r, c| < rad$; radius(r) in C is the line segment $[c, q]$, where q is the intersection between the circumference of C and $[c, r)$. We say that two robots r and r' are on the same radius in C, if $r' \in$ radius(r).

Points in the plane form a *biangular situation* (see Figure 1.a) if there exists a point c (the *center of biangularity*), a polar ordering of the points around c,

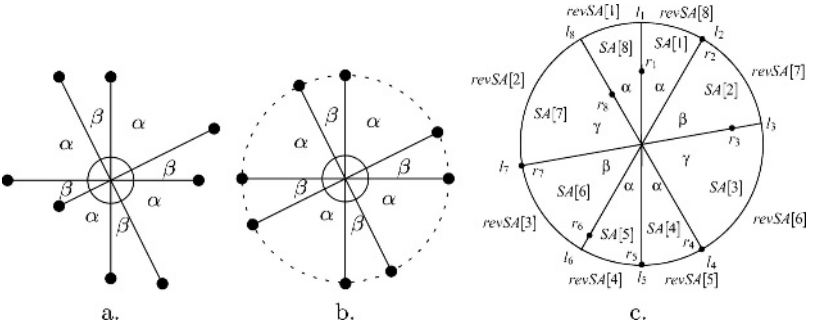


Fig. 1. (a) Biangular situation, (b) Biangular circle; (c) String of angles $\mathrm{SA}(r_1)$

and two nonzero angles α, β such that each two adjacent points form with c angle α or β and the angles alternate.

Points in the plane form a *biangular circle* (see figure 1.b) if they are simultaneously in the biangular situation and on the boundary of a circle with the center of the biangularity equal to the center of the circle.

Given a set of points R in the plane, the *smallest enclosing circle* of the points is the circle with minimal radius such that all points from R are inside or on the circle. We denote the circle by SEC and its center by c, the set R is always unambiguous from the context. The smallest enclosing circle of a set of n points is unique and can be computed in polynomial time[2].

Given a set of points R in the plane, their SEC and c, the *successor* $\mathrm{Succ}(r)$ of any point r is

- either the point $r_i \neq r, r_i \in [c, r]$ with minimal $|r, r_i|$ if such a point exists
- or the point $r_i \neq r$ such that there is no other point inside the angle $\sphericalangle(r, c, r_i)$ and there is no other point on the radius(r_i) further from c

We will denote by $\mathrm{Succ}^{(i)}(r)$ the i-th power of $\mathrm{Succ}(r)$.

Given a set of points R in the plane, the *string of angles* $\mathrm{SA}(r)$ of any point r is the sequence $\{|\sphericalangle(\mathrm{Succ}^{(i-1)}(r), c, \mathrm{Succ}^{(i)}(r))|;\ 1 \leq i \leq n\}$ (see Figure 1c). The *reverse string of angles* $\mathrm{revSA}(r)$ is defined in the same way as string of angles, but all angles are counterclockwise oriented[3] (i.e. $\mathrm{revSA}(r)$ is the reverse of $\mathrm{SA}(r)$). Instead of $\mathrm{SA}(r)$ we will write only SA if we do not consider a specific point r or when the point r is unambiguous from the context. Make a set of all SA and revSA starting in all points and keep only the lexicographically smallest ones. The resulting subset are the *the lexicographically first strings of angles* (LFSA). We will call the size of LFSA the *degree of symmetry* and denote by k.

[2] E.g. take every pair and triple of robots and verify the circle defined by them.

[3] The robots do not have the common sense of clockwise direction, but every individual robot can locally distinguish between a clockwise and counterclockwise orientation.

4 Circle Formation

Problem 1 (Circle formation).
Given is a group of n robots on distinct positions in the plane, arrange them on the boundary of a circle on distinct positions.

We will keep the smallest enclosing circle SEC invariant and use a polar coordinate system based on it. The center of the polar coordinate system is c, the unit of distance *rad*. Angle 0 and clockwise direction are defined only locally for every robot. Angle 0 is either the radius the robot stays on if $c \neq r$, or the robot's local angle 0 otherwise. The clockwise direction is the robot's local one. We will denote $(a, \alpha)_P$ the point at the distance a from c at the angle α (e.g. $(1, 0)_P$ is the intersection of $[c, r)$ and SEC).

Algorithm `Circle_formation` implements the solution: Any robot on SEC stays on its place. All others try to get on SEC along their radii. If more then one robot is on the same radius, only the closest one to SEC will move towards SEC. The others have to wait until they become one of the closest one to c and then do a *side-step*, an atomic move along the concentric circle, to one third of the angle to the next robot. Side-steps are allowed at most in the distance $\frac{1}{2}$ from the center[4]. If a robot wants to do a side-step further than $\frac{1}{2}$ from c, it will move along its radius to this distance first. If the robot is staying on c, it will move anywhere close enough to c to be the nearest to c also in its next cycle.

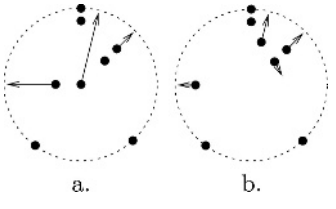


Fig. 2. Circle formation algorithm

4.1 Correctness

Lemma 1. *The smallest enclosing circle SEC is invariant.*

Proof. No robot ever leaves SEC and no robot ever moves behind the boundary of SEC.

Lemma 2. *If a robot did a side-step, there are no other robots on its radius in its new life-cycle and it will visit no occupied radius during its side-step.*

Proof. If a robot does a side-step, other robots on the same radius do not leave it (they cannot be the closest to c) until it finishes the side-step. Robot will occupy

[4] We will use it in chapter 5.

```
CIRCLE_FORMATION(R, r)
 1  d ← Min(|r_i, c|; r_i ∈ R); e ← Min(|r_i, c|; r_i ∈ R, r_i ≠ r)
 2  if (∄r_i ∈ R; r_i ≠ r; r_i ∈ [r, (1, 0)_P])
 3     then return (1, 0)_P
 4  if |r, c| > d
 5     then return r
 6  if r = c
 7     then return (e/2, 0)_P
 8  if |r, c| > 1/2
 9     then return (1/2, 0)_P
10  return (d, |◁(r,c,Succ(r))|/3)_P
```

new radius in its next cycle (Assumption 2). As the robot's new radius is max. in one third to the nearest old occupied radius and so it is for each other robot, no occupied radius was visited and no other robot stays on the destination radius.

Lemma 3. *If a robot chooses the direct way to SEC, it will choose it in all its next life-cycles until it gets there.*

Proof. If a robot r chose the direct way to SEC, there is no other robot on the same radius closer to SEC. Other robots on the same radius cannot cross its way and so is it for all others (Lemma 2).

Lemma 4. *Every robot will reach SEC in finite time.*

Proof. Consider a robot nearest to the c. It will optionally move to the distance $\frac{1}{2}$ and do a side-step. But then it always chooses direct way to SEC and gets there in finite time (Assumptions 1, 2). Eventually, every robot becomes the nearest to c.

5 Biangular Circle Formation

Problem 2 (Biangular circle formation).
Given is a group of n robots on distinct positions, arrange them in a biangular circle pattern.

For at most two robots, the problem is trivially solved – robots can simply stay at their positions. Due to a lack of space the special algorithms for three and four robots are not included. We solve the problem for at least five robots here.

We will keep the smallest enclosing circle SEC invariant and base the polar coordinate system on it in the same way as we did in the circle formation problem. We only override the clockwise direction in some special cases.

5.1 Pseudo-synchronous Circle Model

Imagine the pseudo-synchronous circle model (PSC model) where robots move along the boundary of a fixed circle C. Each step takes finite time and has three phases:

1. Robots synchronously observe the configuration.
2. Robots compute their destination points.
3. Robots asynchronously move to their destinations. Every robot may stop before it reaches its destination (even not move at all). But at least one moving robot (if such a robot exists) must pass a distance greater than some fixed ε or get to the destination.

We call a robot *elected*, if its position at the start of a step is not equal to its computed destination point (i.e. it wants to move). We will denote by $\mathcal{M}(\cdot)$ the set of elected robots.

The algorithm used in computation phase must meet the following rules:

Assumption 3. *Every robot is able to compute* $\mathcal{M}(\cdot)$.

Assumption 4. *It is guaranteed that all robots are at distinct positions during the movement phase.*

Assumption 5. *The degree of symmetry cannot be higher during the robots' movement than it was in the observed configuration. If at least one moving robot did not move, the degree of symmetry must be lower.*

5.2 Emulation of PSC Model in Asynchronous Model

We define the circle C being equal to SEC (SEC must be invariant) and any robot's position as the intersection of its radius and SEC (as the projection on SEC).

Algorithm `Emulate_PSC` implements the emulation. Having observed the configuration, robot calculates:

- The set of elected robots $\mathcal{M}(\cdot)$ (Assumption 3).
- Destination angle $\mathcal{A}(\cdot)$ in PSC model[5].
- Distance $\mathcal{D}(\cdot)$ defined as $\frac{1}{2} + \frac{1}{4k}$ (k is the degree of symmetry).

Default action for any robot r is to move along its radius to SEC (refer to Figure 3). This default action is chosen if any robot is closer to c than $\mathcal{D}(\cdot)$ or if any non-elected robot is inside SEC or if the robot is not elected. Otherwise the robot goes along its radius to the distance $\mathcal{D}(\cdot)$ from c. If and only if all elected robots are at the distance $\mathcal{D}(\cdot)$ from c and all non-elected on SEC, every elected robot does a side-step by angle $\mathcal{A}(\cdot)$.

As long as robots are moving only along their radii, the configuration in the PSC model (the projection on SEC) is invariant and so are the functions $\mathcal{M}(\cdot)$, $\mathcal{A}(\cdot)$ and $\mathcal{D}(\cdot)$. This corresponds to the observation and computation phase of the PSC model.

The algorithm `Emulate_PSC` ensures that all robots are moving only along their radii until all non-elected robots get on SEC and all elected robots get

[5] We emulate a general algorithm in PSC model and $\mathcal{A}(\cdot)$ is the destination in it.

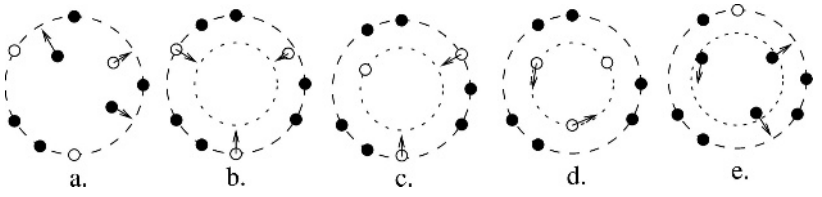


Fig. 3. One step – elected robots are white: (a) nonelected robots are inside SEC; (b) elected robots go onto the dotted circle; (c) waiting for the synchronized configuration; (d) two elected robots observed the synchronized configurations; (e) $\mathcal{D}(\cdot)$ increased (dotted circle grew), new elected robot waits until all robots get out of the dotted circle and all non-elected on SEC

```
EMULATE_PSC(R, r)
 1  if (∃r' ∈ R; |r', c| < D(·))
 2     then return (1, 0)_P
 3  if (∃r' ∈ R \ M(·); r' ∉ SEC)
 4     then return (1, 0)_P
 5  if r ∉ M(·)
 6     then return (1, 0)_P
 7  if (∃r' ∈ M(·); |r', c| ≠ D(·))
 8     then return (D(·), 0)_P
 9  return (D(·), A(r))_P
```

to the distance $\mathcal{D}(\cdot)$ from c. We will call this configuration the *synchronized configuration*. At least one elected robot observes the synchronized configuration and starts doing a side-step. Everything from this moment to the moment a new step starts corresponds to the movement phase of the PSC model.

At the begining of each step the following assumptions must hold:

Assumption 6. *Every moving robot is moving directly towards SEC.*

Assumption 7. *Every robot's distance from c is at least $\mathcal{D}(\cdot)$.*

While the Assumption 7 holds, the observation and computation phases of PSC model are emulated (potentially many times again and again). When the Assumption 7 breaks, the movement phase is emulated. When the Assumption 7 becomes valid again, the movement phase has finished and the next step starts. Note that Assumption 6 cannot be checked by observing the configuration and thus cannot be used in the algorithm.

Correctness of the Emulation. We have to show that we have synchronized the asynchronous robots into the global steps of the PSC model. Robots must not have remembered any movement across the radii when the next step starts, they can be moving only directly towards SEC. In addition we have to show that the next step starts in finite time and assure the progress property of the PSC model.

Lemma 5. *When the next step starts, the Assumption 6 holds.*

Proof. While at least one elected robot did not start doing a side-step, the actual degree of symmetry k is lower (Assumption 5) than the one in the synchronized configuration and thus the actual $\mathcal{D}(\cdot)$ is more than the one in the synchronized configuration.

While at least one robot is doing a side-step, it is closer to c than in the synchronized configuration because robots are moving along a line. Assumption 5 ensures that k do not increase and thus $\mathcal{D}(\cdot)$ will not decrease.

Robots choose the default action whenever at least one robot is closer to c than $\mathcal{D}(\cdot)$. Every elected robot has to either do a side-step (robots that observed the synchronized configuration) and then take the default action until it gets to the distance $\mathcal{D}(\cdot)$ from c or just take the default action until it gets to the distance $\mathcal{D}(\cdot)$. In both cases every elected robot gets to the distance $\mathcal{D}(\cdot)$ only when it cannot have remembered any side-step movement. This ensures that the Assumption 6 holds when the next step starts.

Lemma 6. *The next step starts in a finite time.*

Proof. At the start of every step, non-elected robots get in finite time onto SEC. Then elected robots move in finite time to the distance $\mathcal{D}(\cdot)$ from c. Finally the synchronized configuration is broken and all elected robots get in finite time to the distance at least (actual) $\mathcal{D}(\cdot)$ from c and new step starts.

Lemma 7. *At least one robot (if such robot exists) in PSC model passes every step a distance greater than some fixed ε or get to the destination.*

Proof. At least one elected robot will observe the synchronized configuration and start doing a side-step. Assumption 2 ensures that this robot will pass some minimal distance required by PSC model or get to the destination.

5.3 Idea of the Solution

We will first use the circle formation algorithm to form a circle. Then we switch to the PSC model and try to transform the circle in a finite sequence of steps to the regular one. We will not be able to achieve it in general and will form only the biangular circle in some cases.

Let us denote one string in the set of the lexicographically first strings of angles LFSA by S. Note that the strings in LFSA are the same, they differ only in the starting point and the direction. We have to analyze two cases:

- If all strings in LFSA are oriented in the same direction then only the rotations around c are present. We have the *rotation case* (Figure 4.a) and the string S can be written in form w^k for some w.
- If there are strings in LFSA oriented in clockwise and counterclockwise directions then also the mirrorings about axes passing trough c are present. We have the *mirroring case* (Figures 4.b, 4.c) and the string S can be written in form $(ww^R)^{\frac{k}{2}}$ for some w, where the first and last angles in w and w^R may

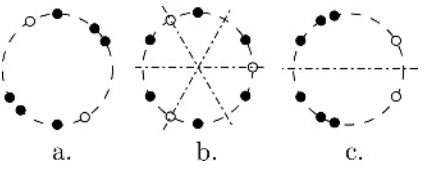


Fig. 4. Undistinguishable (white) robots: (a) rotation case, $k = 2$; (b) mirroring case, $k = 6$; (c) mirroring case, $k = 2$

only be half angles of an angle in SA – when the axis of mirroring symmetry passes trough an axis of an angle. Note that in mirroring case the degree of rotation symmetry is $\frac{k}{2}$.

If all elected robots in PSC model are moving synchronously, the degree of symmetry cannot decrease. We will build the regular pattern in each symmetric part of the circle and will not allow the increase of the degree of symmetry until the regular configuration is achieved. The motivation is simple – the solution must consider this synchronous behavior. When the robots break some symmetry, we start building the regular situation from the start with lower degree of symmetry. As the degree of symmetry is a natural number, the process will restart only finite number times. The only exception is the increased degree of symmetry in case the regular circle formation is reached[6].

We have to solve many technical details: We are going to concatenate the circle formation algorithm and the PSC emulation algorithm. We have to perform steps in the PSC model in order to form the regular circle in finite time and assure the invariance of SEC and take care to not increase the degree of symmetry.

5.4 Concatenation of Circle Formation and PSC Model

The concatenation of two different algorithms is not possible in general. We have to construct new algorithm, which chooses and calls the right subalgorithm using only the observed configuration.

In our case everything is prepared and the concatenation is done very simply: the circle formation algorithm is used when at least two robots occupy the same radius, the PSC model emulation otherwise.

Correctness of the Concatenation. If all robots occupy different radii at the start, only the PSC model emulation algorithm will be used. Otherwise the circle formation algorithm is used first until the last two robots on the same radius are separated. This is done only by robot(s) doing side-steps in the distance not more than $\frac{1}{2}$ from c. This is correct start for the PSC model emulation algorithm because only default actions will be performed until the robots finish their side-steps and move along their radii close enough towards SEC to start the first step.

[6] Regular circle will not be formed only when robots are in the moment the circle formation algorithm has finished in the biangular configuration.

5.5 Choosing Elected Robots in PSC Model

Assumption 3 gives us the first restriction. In addition we will elect only the smallest possible number of robots.

Lemma 8. *The minimal size of a nonempty subset of robots that all robots can agree on from the strings of angles is equal either to* |LFSA| *if no robot is on an axis of a mirroring symmetry, or to* $\frac{|\mathrm{LFSA}|}{2}$ *otherwise.*

Proof. The robot r must recognize whether it is elected or not only from its set $\{\mathrm{SA}(r),\ \mathrm{revSA}(r)\}$, because all other robots must agree with it and the other robots do not know its local clockwise orientation.

The robots with equal $\{\mathrm{SA}, \mathrm{revSA}\}$ must be all elected or not, because the robots run the same deterministic algorithm. Thus if a robot r is elected, all other robots with equal SA or revSA must be elected too; $|\{\mathrm{SA}|\ \mathrm{SA} = \mathrm{SA}(r)\}| = |\mathrm{LFSA}|$. Thus the minimal size of a nonempty subset of robots that all robots can agree on is either |LFSA| if $\mathrm{SA}(r) \neq \mathrm{revSA}(r)$, or $\frac{|\mathrm{LFSA}|}{2}$ otherwise.

According to Assumption 3 and Lemma 8 we have to elect one robot for each period w in the rotation case and two symmetric robots (one in the special case) for each rotation period ww^R in the mirroring case. It is therefore sufficient to define the elected robots $\mathcal{M}(\cdot)$ and their moves $\mathcal{A}(\cdot)$ only for the smallest rotation period and this implicitly defines this movement for the whole configuration.

In a biangular configuration (regular being a special case) all robots would be elected and the use of the PSC model emulation would lead to breaking the invariance of SEC. This is why biangular (regular) configurations are handled as a special case – no robot is elected and applying the default action robots move onto SEC and stay there.

5.6 Critical Robots

Invariance of SEC during one step is easily achieved when at least one non-identical rotation is among the symmetries (Figures 4.a,b). Any not moving undistinguishable robots on SEC assure its invariance. Otherwise, we cannot send arbitrary robot into SEC without breaking its invariance (Figure 5.b).

We will call a (set of) robot(s) *critical* if its deletion from SEC modifies SEC (see Figures 5.a,c,d, 4.c).

Lemma 9. *A robot is critical if and only if the sum of its two adjacent angles is greater than* $180°$*. If* $n \geq 4$*, at most two robots can be critical and the critical robots are neighbors.*

Proof. Three (two) robots on SEC forming a triangle (line) such that c is inside this triangle (line) are sufficient to assure the invariance of SEC. The removal of a robot with the sum of adjacent angles at most $180°$ thus cannot modify SEC.

If there are two not neighboring critical robots, the sum of their adjacent angles would be more than $360°$: $2(180° + \varepsilon) > 360°$.

Three neighboring critical robots make similar contradiction (see Figure 5.e): $\alpha + \beta + \gamma + \delta > x + (180° - x) + x + (180° - x) = 360°$.

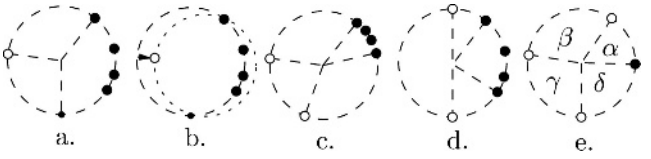


Fig. 5. Critical (white) robots cannot leave SEC. (a) one critical robot; (b) critical robot breaking the invariance of SEC; (c) two critical robots; (d) two critical robots form angle 180°; (e) three robots cannot be critical

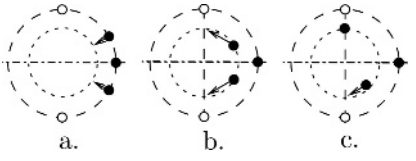


Fig. 6. 180° angle is handled as a special case. Critical robots are white. (a) elected robots going to synchronize in distance $\mathcal{D}(\cdot)$; (b) elected robots move to the radii of critical robots; (c) all robots wait until the robot doing side-step finishes

Consider a configuration where two critical robots form with c angle 180° (see Figure 5.c). These two critical robots cannot move. We forbade in the PSC model the robots to stay on the same place and thus no other robot can ever get between the critical robots. We must make an exception.

The nearest robot(s) to the critical ones (with lex. smaller $\{\mathrm{SA}, \mathrm{revSA}\}$, both for equal) will move onto the critical robot(s) in the PSC model[7] (Figure 6). When it (they) get there, circle formation algorithms will be activated and it (they) move between the two critical robots. We achieve it by defining the clockwise direction of the moving robot(s) in the direction to the 180° angle.

Correctness of the Exception. The two critical robots assure the invariance of SEC. The nearest robot(s) to the critical ones is(are) moving onto it(them), thus it(they) will be the nearest also in the next step. Following Lemma 7 we get that it(they) reach(es) the critical robot(s) in finite time.

The switch to the circle formation algorithm is done in the moment, when the first moving robot r_1 reached the destination and the other one r_2 (if exist) may be still moving (see figure 6.c). In such a case r_1 and all other robots will wait (because r_2 is strictly closer to c than any other robot) until r_2 either reaches its destination or wakes up and goes directly towards SEC. In all cases we have a correct starting situation for the circle formation algorithm. One another robot ($n \geq 5$) always stays on SEC and it ensures that the 180° angle disappears and robots will continue as if there never was any 180° angle.

[7] No robots will meet in reality, only in the projection in PSC model.

5.7 Rotation Case

For a given period of angles w we must elect one robot and move it by an angle in order to convert w into the form $\alpha^{|w|}; \alpha = \frac{360^\circ}{kn}$ in finite time. As we do not have to break any symmetries, we can define the first matching robot as the one with the smallest $\{\mathrm{SA}, \mathrm{revSA}\}$.

For our construction we need the configuration with exactly one maximal angle. If this is not the case, we move the first noncritical robot at one side of one of the greatest angles in order to enlarge one maximal angle (Figure 7.a). We will let it pass an angle small enough to not (potentially) increase the degree of symmetry.

Now we know that one angle (mark as τ) in w is the strictly greatest. We will take care to hold the size of all other angles bellow τ. If we choose in the following to move a robot in such a way that τ could become not the strictly greatest, we move the next one first.

One strictly greatest angle ensures that no rotation symmetry can arise. The only symmetry we will have take care for is the mirroring about the axis running trough the axis of τ (later only *axis*) (see Figure 7.b).

We will build a sequence of α angles starting at one side of τ. If the sequence of α angles is not complete on either of the axis' sides, we first complete one side. We choose the side of the axis with more robots. If equal number of robots is on both sides, we either move the robot on the axis to one side, either move one robot (closest to the axis or any) onto the axis (see Figure 7.b).

The different number of robots on the sides of the axis ensures that the mirroring symmetry about the axis cannot arise while robots are moving only on one side of the axis. So we can build a sequence of α angles on one of the axis and cram all needless robots between the axis and the last robot in the sequence (see Figures 7.c,7.d). The lengthening of the sequence by one angle is as follows: cram all the abundant robots between the place for the new robot in the sequence and first robot next to this place (b.e. in the middle) or spread them in order to keep all angles bellow τ. Then move the new robot on its place in the sequence.

When the sequence is complete on one side, we continue in building the sequence on the other side (see Figures 7.d-f). We must take care for the mirroring. If the chosen robot's move would increase the degree of symmetry, we first move the next robot a bit (b.e. halfway or less between the original destina-

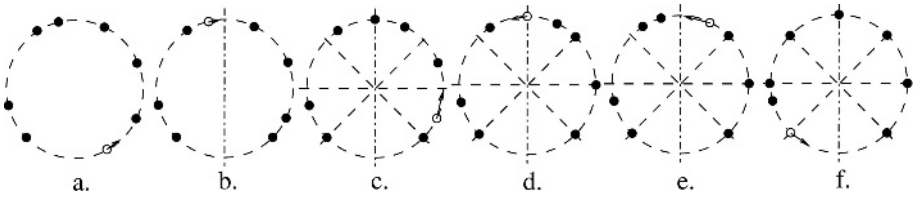


Fig. 7. Rotation case. (a) creating one strictly greatest angle; (b) assuring different number of robots on the sides of "axis"; (c)-(f) building a sequence of α angles

tion and next robot's position). At the end the robots stay in the regular circle pattern.

For the correctness observe: the degree of symmetry could not rise until the regular circle pattern is formed, SEC remained invariant, regular circle pattern is formed in finite time.

5.8 Mirroring Case

For each string of angles ww^R in a mirroring configuration two symmetric robots (or one on the axis of mirroring) must be elected and the angle they move by must be defined.

The representation ww^R of the period is not sufficient, we also need to know whether there are robots at the beginning and end of w (i.e. whether the robots stay on the intersections of SEC and the axis of mirroring).

When a robot is an the axis of mirroring, we will break the mirroring symmetry and convert the problem to the rotation case in the lower degree of symmetry. The robot on the axis with lex. smaller $\{\mathrm{SA}, \mathrm{revSA}\}$ will move to any side of the axis by an angle one third of the distance to the next robot (see Figure 8a). If this move could rise the degree of symmetry, the robot will move by a smaller angle or to the other side. The side cannot be chosen globally by all robots, but the moving robot can use its local clockwise direction.

So it is sufficient to consider the cases with no robots on the axis of mirroring (and thus $n \geq 6$). Unlike the rotation case, in the mirroring case we cannot start building the sequence of α angles anywhere. We must align the sequence of robots to fit the mirroring symmetry. The nearest robot by the axis will go to the angle $\frac{\alpha}{2}$ from the axis.

To avoid the increasing of the degree of symmetry, we create (in analog to the rotation case) one strictly greatest angle and take care to never increase any other angle to its size. The adjacent half-angles about the axis are considered as one angle.

We will build the sequence of α angles from both sides of w. We will lengthen the sequence at that side, where the lengthening will not break the strictly greatest angle and the critical robots need not move. If the sequence can be lengthened on both sides, we continue in the one where the robot dedicated to move is closer to its target. The process of lengthening is the same as in the rotation case.

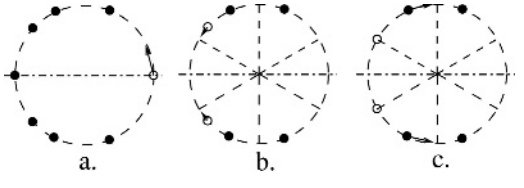


Fig. 8. Forming the regular circle in the mirroring case. (a) robot on the axis is breaking the mirroring symmetry; (b),(c) building a sequence of aligned α angles

6 Conclusions and Open Issues

We studied in this paper the problem of forming the biangular circle formation. First we presented solution for an easy problem – circle formation. Then we showed that robots can in finite time rearrange the circle to the biangular circle.

The regular circle is not formed only in cases when the biangular configuration is observed in the PSC model. Note that if the robots would have a common sense for clockwise orientation, the electing function $\mathcal{M}(\cdot)$ could be based just on clockwise strings of angles SA. Robots would be able to elect nontrivial subset of them in all cases (except regular circle) and continue in forming the regular circle. This single modification of the algorithm solves the regular circle formation problem in the model with the common sense of clockwise orientation.

Open questions:

- Is it possible to form a regular circle in general case? Try to characterize the configurations transformable into the regular circle.
- Is it possible to form regular circle in the pseudosynchronous model?
- The algorithm supposes infinite precision in the observation. Is it possible to eliminate it? How to define such a model?
- Time complexity of the solution may be defined, analyzed and improved.

Acknowledgement. Author would like to thank Rastislav Královič, Dana Pardubská and Michal Forišek for their support.

References

1. X.Défago, A. Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In Proc. of the 2nd ACM Annual Workshop on Principles of Mobile Computing (POMC'02), pages 97-104, Toulouse, France, October 2002
2. Cieliebak, P. Flocchini, G. Prencipe and N. Santoro. Solving the Robots Gathering Problem. In 30th International Colloquium on Automata, Languages and Programming (ICALP 2003), to appear. Eindhoven, The Netherlands, 30 Giugno – 4 Luglio, 2003.
3. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Asynchronous Mobile Robots with Limited Visibility. In STACS, 2001.
4. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In ISAAC '99, pages 93-102, 1999
5. I. Chatzigiannakis, M. Marcou, S. Nikoletseas: Distributed circle formation for anonymous oblivious robots. WEA 2004
6. Noa Agmon, David Peleg: Fault-tolerant gathering algorithms for autonomous mobile robots. SODA 2004: 1070-1078
7. Ichiro Suzuki, Masafumi Yamashita: Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. SIAM J. Comput. 28(4): 1347-1363 (1999)

Hardness and Approximation Results for Black Hole Search in Arbitrary Graphs*

Ralf Klasing**, Euripides Markou***, Tomasz Radzik†, and Fabiano Sarracco‡

Abstract. A black hole is a highly harmful stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. We consider the task of locating a black hole in a (partially) synchronous arbitrary network, assuming an upper bound on the time of any edge traversal by an agent. For a given graph and a given starting node we are interested in finding the fastest possible Black Hole Search by two agents (the minimum number of agents capable to identify a black hole). We prove that this problem is NP-hard in arbitrary graphs, thus solving an open problem stated in [2]. We also give a 7/2-approximation algorithm, thus improving on the 4-approximation scheme observed in [2]. Our approach is to explore the given input graph via some spanning tree. Even if it represents a very natural technique, we prove that this approach cannot achieve an approximation ratio better than 3/2.

Keywords: approximation algorithm, black hole search, graph exploration, mobile agent, NP-hardness.

1 Introduction

Problems related to security in a network environment have attracted many researchers. For instance protecting a host, i.e., a node of a network, from an agent's attack [11, 12] as well as protecting mobile agents from "host attacks",

* Research supported in part by the European project IST FET CRESCCO (contract no. IST-2001-33135), the Royal Society Grant ESEP 16244, EGIDE, and the Ambassade de France en Grèce/Institut Français d' Athènes. Part of this work was done while E. Markou, T. Radzik and F. Sarracco were visiting the MASCOTTE project at INRIA Sophia Antipolis.

** MASCOTTE project, I3S-CNRS/INRIA/Université de Nice-Sophia Antipolis, 2004 Route des Lucioles, BP 93, F-06902 Sophia Antipolis Cedex (France), email Ralf.Klasing@sophia.inria.fr

*** Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, email emarkou@softlab.ece.ntua.gr

† Department of Computer Science, King's College London, London, UK, email radzik@dcs.kcl.ac.uk

‡ Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", email Fabiano.Sarracco@dis.uniroma1.it

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 200–215, 2005.

i.e., harmful items stored in nodes of the network, are important with respect to security of a network environment. Various methods of protecting mobile agents against malicious hosts have been discussed, e.g., in [8, 9, 10, 11, 12, 13].

We consider here malicious hosts of a particularly harmful nature, called *black holes* [1, 2, 3, 4, 5, 6]. A black hole is a stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. We are dealing with the issue of locating a black hole: assuming that there is at most one black hole in the network, at least one surviving agent must find the location of the black hole if it exists, or answer that there is no black hole, otherwise. The only way to locate the black hole is to visit it by at least one agent, hence, as observed in [4], at least two agents are necessary for one of them to locate the black hole and survive. Throughout the paper we assume that the number of agents is the minimum possible for our task, i.e., 2, and that they start from the same node, known to be safe.

The issue of efficient black hole search was extensively studied in [3, 4, 5, 6] in many types of networks under the scenario of a totally asynchronous network (i.e., no upper bound on this time needed for an edge traversal). In this setting it was observed that, in order to solve the problem, the network must be 2-connected. Moreover, it is impossible to answer the question of whether a black hole actually exists in an asynchronous network, hence [3, 4, 5, 6] work under the assumption that there is exactly one black hole and the task is to locate it.

In [1, 2] the problem is studied under the scenario we consider in this paper as well. The network is partially synchronous, i.e. there is an upper bound on the time needed by an agent for traversing any edge. This assumption makes a dramatic change to the problem: the black hole can be located by two agents in any graph; moreover the agents can decide if there is a black hole or not in the network. If, without loss of generality, we normalize to 1 the upper bound on edge traversal time, then we can define the cost of a Black Hole Search as the time taken under the worst-case location of the black hole (or when it does not exist in the network), assuming that all the edge traversals take time 1. In [2] the BHS problem is studied in tree topologies, while in [1] a variant of the problem is studied in which the black hole can be located only in a given set of nodes (which is a subset of the set of nodes of the graph) and it is proved that this variant is NP-hard in an arbitrary graph.

In this paper we show that the problem of finding the minimum cost Black Hole Search by two agents in an arbitrary graph is NP-hard even under the restricted scenario of one safe node (the starting node), thus solving an open problem stated in [2]. Moreover, we give a 7/2-approximation algorithm for this problem, i.e., we construct a polynomial time algorithm which, given a graph and a starting node as input, produces a Black Hole Search whose cost is at most 7/2 times larger than the best Black Hole Search for this input. This result improves on the 4-approximation scheme observed by [2]. Finally, we show that any Black Hole Search that explores the given input graph via some spanning tree cannot have an approximation ratio better than 3/2.

2 Model and Terminology

Let $G = (V, E)$ be a connected graph. We assume that the nodes of G can be partitioned into two subsets: a set of BLACK HOLES $B \subsetneq V$, i.e. nodes destroying any agent visiting them without leaving any trace; and a set of SAFE nodes $V \setminus B$. During a Black Hole Search (or simply BHS), a set of agents starts from a special node $s \in V \setminus B$ (which we call STARTING NODE), and explores the graph G by traversing its edges. Obviously s is known to be a safe node; more generally, there exists a subset $\widehat{S} \subseteq V \setminus B$ of nodes initially known to be safe. The target of the agents is to report to s the information on which nodes of G are black holes.

In this paper we consider the following restricted version of the problem: $|B| \leq 1$ (i.e. there can be either one black hole or no black holes at all in G), $\widehat{S} = \{s\}$ (only the starting node is initially known to be safe), the set of agents has size 2, agents have a complete map of G, agents have distinct labels (we will call them *Agent*-1 and *Agent*-2) and they can communicate only when they are in the same node (and not, e.g., by leaving messages at nodes). Finally, the network is (at least partially) synchronous. We consider the following formalization, called the *Minimum Cost BHS Problem*, or simply BHS problem.

Instance : a graph $G = (V, E)$, and a node $s \in V$.
Solution : an EXPLORATION SCHEME $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$ for G and s, i.e. two equal-size sequences of nodes in G, $\mathbb{X} = \langle x_0, x_1, \ldots, x_T \rangle$ and $\mathbb{Y} = \langle y_0, y_1, \ldots, y_T \rangle$ which satisfy the feasibility constraints listed below.
Measure : the cost of the BHS based on $\mathcal{E}$.

When a BHS based of $\mathcal{E}$ is performed in G, *Agent*-1 follows the path defined by $\mathbb{X}$ while *Agent*-2 follows the path defined by $\mathbb{Y}$, in synchronized steps. In other words, at the end of the i-th step of the exploration scheme, *Agent*-1 is in node x_i, while *Agent*-2 is in node y_i. We say that each step has length one time unit. As soon as an agent deduces the existence and the exact location of the black hole, it "aborts" the exploration and returns to the starting node s by traversing nodes in $V \setminus B$. A pair of sequences of nodes $\mathbb{X}$ and $\mathbb{Y}$ defines a feasible exploration scheme for a graph G and a starting node s, which can be effectively used as a basis for a BHS on G, if it satisfies the following four constraints.

Constraint 1: $x_0 = y_0 = s$, $x_T = y_T$.
Constraint 2: for each $i = 0, \ldots, T-1$, either $x_{i+1} = x_i$, or $(x_i, x_{i+1}) \in E$; similarly for *Agent*-2, either $y_{i+1} = y_i$ or $(y_i, y_{i+1}) \in E$.
Constraint 3: $\bigcup_{i=0}^{T} \{x_i\} \cup \bigcup_{i=0}^{T} \{y_i\} = V$.

We need some further definitions to state the fourth constraint. Given an exploration scheme $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$, the EXPLORED TERRITORY at step i is

$$S_i = \begin{cases} \bigcup_{j=0}^{i} \{x_j\} \cup \bigcup_{j=0}^{i} \{y_j\}, & \text{if } x_i = y_i; \\ S_{i-1}, & \text{otherwise.} \end{cases}$$

Observe that, by Constraint 1, $S_0 = \{s\}$ and, by Constraint 3, $S_T = V$. A node v is EXPLORED at step i if $v \in S_i$, otherwise it is UNEXPLORED. The definition of explored territory covers the assumption that, whenever the two agents are in the same node, they communicate to each other that the nodes of the network they visited are safe. A MEETING STEP (or simply MEETING) is the step 0 and every step $1 \leq j \leq T$ such that $S_j \supsetneq S_{j-1}$. Observe that for each meeting step j, we must have $x_j = y_j$ (but not necessarily the opposite); we call this node a MEETING POINT. Each sequence of steps $\langle j+1, j+2, \ldots, k\rangle$ between two consecutive meetings j and k is a PHASE of length $k-j$. Now we can give the last constraint for a feasible exploration scheme. It says that during each phase, an agent can visit at most one unexplored node, and the same unexplored node cannot be visited by both agents (see [2]).

Constraint 4: for each phase $\langle j+1, \ldots, k\rangle$,
$|\{x_{j+1}, \ldots, x_k\} \setminus S_j| \leq 1$, $|\{y_{j+1}, \ldots, y_k\} \setminus S_j| \leq 1$, and
$\{x_{j+1}, \ldots, x_k\} \setminus S_j \neq \{y_{j+1}, \ldots, y_k\} \setminus S_j$.

Lemma 1. *If k is a meeting step of exploration scheme $\mathcal{E}$, then $x_k = y_k \in S_{k-1}$. Each phase of $\mathcal{E}$ has length at least two.*[1]

Any length-2 phase $\langle j+1, j+2\rangle$ at the end of which the explored territory increases by 2 nodes must have the following structure. Let m be the meeting point at step j. During step $j+1$, *Agent-i* visits an unexplored node v_i adjacent to m. In step $j+2$, the agents meet in a safe node adjacent to both v_1 and v_2. Note that this node can be either m, and in this case we denote the phase as *b-split*(m, v_1, v_2), or a distinct node m', and in this case we denote the phase as *a-split*(m, v_1, v_2, m').

For an exploration scheme $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$ and a "location" of a black hole B, where $B = \emptyset$ or $B = \{b\}$ for $b \in V \setminus \{s\}$, the EXECUTION TIME is defined as follows. If $B = \emptyset$, then the execution time is equal to the length T of the exploration scheme, plus the shortest path distance from $x_T (= y_T)$ to s. In this case the agents must perform the full exploration, and then get back to s. If $B = \{b\}$, then let j be the first step in $\mathcal{E}$ such that $b \in S_j$. The execution time in this case is equal to j plus the shortest path distance from $x_j (= y_j)$ to s avoiding b. One agent, say *Agent*-1, vanishes into the black hole b during the phase ending at step j, so it cannot meet *Agent*-2 at the meeting point $x_j = y_j$. Thus the surviving *Agent*-2 knows at the end of step j the exact location of the black hole (see Constraint 4), so it can go straight back to s. The COST of the BHS based on an exploration scheme $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$ (or simply the COST of $\mathcal{E}$) is the maximum of the execution times of $\mathcal{E}$ for all possible locations of the black hole B.

It is easy to check that if G is a tree, then the case $B = \emptyset$ gives always the maximum execution time among all possible locations of the black hole in the nodes of G. If G is an arbitrary graph, then this property does not always hold.

[1] Some proofs are omitted due to the space restrictions.

3 NP-Hardness of the BHS Problem in Arbitrary Graphs

In this section, we prove the NP-hardness of the BHS problem in arbitrary graphs by providing a reduction from a particular version of the Hamiltonian Circuit problem to the decision version of the BHS problem.

cpHC problem

Instance : cubic planar graph $G = (V, E)$, and an edge $(x, y) \in E$;
Question : does G contain a Hamiltonian cycle that includes edge (x, y)?

dBHS problem

Instance : graph $G' = (V', E')$, with a starting node $s \in V'$, and a positive integer X;
Question : does there exist an exploration scheme $\mathcal{E}$ for G' starting from s, such that the BHS based on $\mathcal{E}$ has cost at most X?

One can check that the reduction from the 3-SAT problem to the Hamiltonian Cycle problem given in [7] proves actually that the cpHC problem is NP-hard. For an arbitrary instance of the cpHC problem (i.e. for each planar cubic graph $G = (V, E)$ and edge $(x, y) \in E$), we construct in linear time a corresponding instance of the dBHS problem (i.e. a graph G', a starting node s, and an integer X) such that the original instance is a positive instance of the cpHC problem if and only if the constructed instance is a positive instance of the dBHS problem.

Since G is planar, we can find in linear time an (arbitrary) combinatorial planar embedding of G, i.e. a clockwise order L_v of the neighbors of each node $v \in V$. We then construct both G' and its embedding, as extensions of G and its (combinatorial planar) embedding, in the following five steps.

1. G' has originally the same nodes (*original nodes*), the same edges and the same embedding as G.
2. Replace the edge (x, y) with the edges (x, s) and (s, y), where $s \notin V$ is a new node. The node s replaces the node y in L_x, and the node x in L_y.
3. For each edge (v, w) in the current graph (the current set of edges is $E \cup \{(x, s), (s, y)\} \setminus \{(x, y)\}$) add two nodes $z_1^{(v,w)}$ and $z_2^{(v,w)}$ (twin nodes) and four edges $(z_1^{(v,w)}, v)$, $(z_1^{(v,w)}, w)$, $(z_2^{(v,w)}, v)$ and $(z_2^{(v,w)}, w)$. For the embedding, place $z_1^{(v,w)}$ before and $z_2^{(v,w)}$ after w in L_v. Similarly, place $z_1^{(v,w)}$ after and $z_2^{(v,w)}$ before v in L_w.
4. For each node $v \in V \cup \{s\}$ and for each pair of nodes $z_i^{(v,w)}$, $z_j^{(v,u)}$ consecutive in L_v, add an edge between these $z_i^{(v,w)}$ and $z_j^{(v,u)}$. We call this edge a *shortcut edge*. Let $z_j^{(v,u)}$ follow $z_i^{(v,w)}$ in L_v. Then, in the embedding of G', place $z_j^{(v,u)}$ immediately before v in the order of the neighbors of $z_i^{(v,w)}$, and place $z_i^{(v,w)}$ immediately after v in the order of the neighbors of $z_j^{(v,u)}$.

5. For each node $v \in V \cup \{s\} \setminus x$, add a new node v^F (*flag node*) and an edge (v, v^F). For the embedding, v^F can be in any place in L_v.

Figure 1, which will be used as an illustration for the proof of Lemma 5, illustrates also the construction of graph G'. If $n = |V|$ and $e = |E|$, are the numbers of nodes and edges in G, then graph G' has n original nodes, one starting node s, n flag nodes and $2(e + 1)$ twin nodes. Since in cubic graphs $e = \frac{3}{2}n$, the total number of nodes in G' is $5n+3$. This construction can be done in linear time with respect to the size of G. We assume that s is the starting node, while the remaining $5n + 2$ nodes are initially unexplored. We set $X = 5n + 2$.

Lemma 2. *If u and w are two original nodes having a common neighbor v in G, then there exists in G' a path $[u, z', z'', w]$ where z' is a twin node for the edge (u, v) and z'' is a twin node for the edge (v, w).*

Lemma 3. *Each twin node in G' has degree 4.*

Lemma 4. *If the graph G has a Hamiltonian cycle that includes edge (x, y), then there exists an exploration scheme $\mathcal{E}_{HC}$ on G' starting from s, such that the BHS based on it has cost at most $5n + 2$.*

Proof. Let $\{v_1 = y, e_1, v_2, \ldots, e_{n-1}, v_n = x, e_n, v_1 = y\}$ be such Hamiltonian cycle in G. Consider the exploration scheme $\mathcal{E}_{HC}$ defined by the following sequence of phases:

1. **b-split**(s, s^F, y), where s^F is the flag node of s;
2. **a-split**(s, z_1, z_2, y), where z_1 and z_2 are the twin nodes of the edge (s, y);
3. for each node v_i of the Hamiltonian cycle, with $(i = 1, \ldots, n - 1)$:
   (a) let v_j be the third neighbor of v_i, other than v_{i-1} and v_{i+1}; if $j > i$ then **b-split**(v_i, z_1, z_2), where z_1 and z_2 are the twin nodes of (v_i, v_j);
   (b) **b-split**(v_i, v_i^F, v_{i+1}), where v_i^F is the flag of v_i;
   (c) **a-split**(v_i, z_1, z_2, v_{i+1}), where z_1 and z_2 are the twin nodes of the edge (v_i, v_{i+1});
4. **a-split**(x, z_1, z_2, s), where z_1 and z_2 are the twin nodes of the edge (x, s).

Now let us compute the length of $\mathcal{E}_{HC}$. As we have seen in Section 2, each *a-split* and *b-split* phase has length 2, and increases the explored territory by 2 nodes. The overall number of phases is therefore $(5n + 2)/2$ and hence $\mathcal{E}_{HC}$ has length $5n + 2$. Notice that this is also the exploration time of $\mathcal{E}_{HC}$, for the case $B = \emptyset$, since $\mathcal{E}_{HC}$ ends in s.

Claim. Consider the meeting step when the agents are to meet at a node v_i. If a black hole has been just discovered, then the remaining cost of the BHS is not greater than the remaining cost in the case of no black hole.

Proof. (Sketch) It suffices to show that if there is a black hole in G', then the surviving agent can keep following the Hamiltonian Cycle (possibly by using a shortcut edge), and get to s in less time units than in the case $B = \emptyset$.

This implies that also the cost of the BHS based on $\mathcal{E}_{HC}$ is $5n + 2$, i.e. there is no allocation of the black hole that yields a larger exploration time. Observe that the BHS defined above is optimal since it is not possible to explore $5n + 2$ nodes in less that $5n + 2$ time units. □

Lemma 5. *If there exists an exploration scheme on G' starting from s such that the cost of the BHS based on it has cost at most $5n + 2$, then the graph G has a Hamiltonian cycle that includes edge (x, y).*

Proof. Let $\mathcal{E}_\sigma$ be such exploration scheme. By Lemma 1, each phase of $\mathcal{E}_\sigma$ has length at least two and cannot explore more than two unexplored nodes. Since G' has $5n + 2$ unexplored nodes, $\mathcal{E}_\sigma$ must end in s, and each of its phases must be either an *a-split* or a *b-split*. Consider the sequence M_σ of the meeting points established for $\mathcal{E}_\sigma$ at the end of each *a-split*, excluding the last one which is s.

Claim. Nodes x and y must be the two endpoints of M_σ and s cannot be in M_σ.

Each meeting point v_i in M_σ other than s must have at least degree 5 since one neighbor is needed for the initial exploration of v_i, two unexplored neighbors are needed for the *a-split* that ends in v_i and two further unexplored neighbors are needed for the *a-split* that leaves v_i. For this reason only the original nodes of G' (neither flags nor twins) can be in M_σ. Finally, each flag node has to be explored with a *b-split* having as meeting point the original node adjacent to it, hence each original node of G' (i.e. each node of G) must be in M_σ. Now we prove that the sequence M_σ defines a Hamiltonian cycle on G, i.e.:

a) each node of G appears exactly once in M_σ;

b) if nodes v_i and v_j are consecutive in M_σ, then the edge (v_i, v_j) must be in G.

We start by proving *a)*. We have seen that each node of G is in M_σ, thus we have to prove that no node appears twice or more. Compute the neighbors needed by a node v_i in M_σ: at least one neighbor is needed for the initial exploration of v_i (two neighbors, if it is done through an *a-split*). Then, for each occurrence of v_i in M_σ, two unexplored neighbors are needed for meeting in v_i with an *a-split*, and two additional unexplored neighbors are needed for leaving v_i with an *a-split*. Moreover the flag v_i^F has to be explored with a *b-split* from v_i, hence another unexplored neighbor of v_i is needed. If the node v_i occurs k times in M_σ, then the total number of neighbors needed by v_i is at least $1 + 4k + 2 = 3 + 4k$. Since each original node in G' has only 10 neighbors (as G is a cubic graph), it must be $k \leq 1$, thus each node appears exactly once in M_σ.

Now we prove property *b)* of M_σ. According to the structure of G', *a-split* operations can either explore two twin nodes of an original edge (in this case property *b)* is verified since the meeting point is adjacent in G to the previous one), or explore two original nodes of G' and meet in another original node which may not be adjacent to the previous meeting point, thus violating property *b)*. Suppose that this latter kind of split (a *big a-split*) happens from a node A to a node B; see Figure 1. In order to do this, A must have two unexplored original

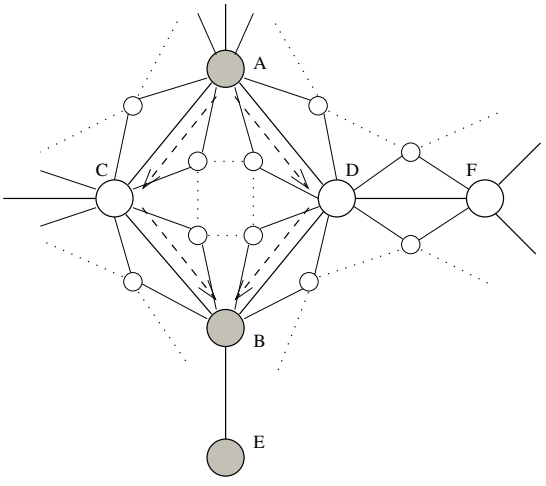


Fig. 1. A big *a-split* from A to B. Big circles – original nodes in G; small circles – twin nodes; shortcut edges are dotted; flag nodes are not represented

neighbors (C and D in the figure) both having B as a neighbor. B must be already explored, therefore the last original neighbor of B (E in the figure) must have already been a meeting point (we can suppose without loss of generality that the one from A to B is the first *big a-split* in M_σ). At this point no other *big a-splits* can be performed from B (all its original neighbors are now explored) and, by property *a)*, E cannot be again a meeting point, thus the sequence M_σ can have either C or D as the next meeting point. Supposing that C is that one, consider the instant when D becomes a meeting point. We cannot get to D with a *big a-split*, since D does not have two neighbors in G that are unexplored, hence also F has been already a meeting point. Now all the original neighbors of D have already been a meeting point in M_σ, and none of them can be s, thus there is no way to leave D without violating property *a)*. Therefore there cannot be any *big a-split* in σ, and thus also property *b)* is verified. □

4 An Approximation Algorithm for the BHS Problem

One may approach the BHS problem in an arbitrary graph G in the following way. First select a spanning tree in G and then search the graph using the tree edges. As observed in [2], this approach guarantees an approximation ratio of 4 since the following exploration of an n-node tree requires at most $4(n-1)$ steps. Both agents traverse the tree together in, say, the depth-first order and explore each new node v with a two-step *probe phase*: one agent waits in the parent p of v while the other goes to v and back to p.

To follow this "spanning-tree" approach effectively, we need good exploration schemes for trees and good spanning trees for those schemes. Intuitively a good heuristic for the former problem should be to minimize the time spent by one

agent waiting for the other one. A good heuristic for the latter problem should be to minimize the number of nodes without siblings in the selected spanning tree. It may be difficult, if possible at all, to schedule exploration of such nodes not using probe phases (which imply waiting).

We assume throughout this section that the starting node s has degree at least 2. In Sections 4.1 and 4.2 we present algorithms *Search-Tree*(T, s) and *Generate-Tree*(G, s), which implement the above general heuristics. Algorithm *Search-Tree*(T, s) generalizes the algorithm proposed in [2] for so-called *bushy trees*. The *Spanning Tree Exploration (STE)* algorithm returns for G and s the exploration scheme computed by *Search-Tree*(T_G, s), where T_G is the spanning tree computed by *Generate-Tree*(G, s). In Section 4.3 we show that the approximation ratio of the *STE* algorithm is at most 7/2.

4.1 Exploration Schemes for Trees

Let T be a rooted n-node tree and let s denote its root. Our algorithm *Search-Tree*(T, s) for constructing an exploration scheme for T uses the following order $L(T)$ of the nodes of T other than the root. We first order the children of each node according to the number of descendants: a child with more descendants comes before a child with fewer descendants and the ties are resolved arbitrarily. Thus from now on T is an *ordered* rooted tree. Let $\langle w_1, w_2, \ldots, w_p \rangle$ be the sequence of the internal nodes of T ordered according to their depth-first-search numbers. The order $L(T)$ is this sequence with each node w_i replaced by the (ordered) list of its children. The i-th node in the order $L(T)$ will be denoted by v_i and called the i-th node of the tree. The odd (even) nodes of T are the nodes at the odd (even) positions in $L(T)$.

We classify all nodes other than the root s into the following three types. The *type-1* nodes are the leaves of T; the *type-3* nodes are the internal nodes with at least one sibling; and the *type-4* nodes are the internal nodes (other than the root) without siblings. Informally, in the exploration scheme which we produce for T a *type-i* node contributes i units to the total cost. Note that there is no type 2. We denote by x_t the number of *type-t* nodes. We consider first the case when T does not have any *type-4* nodes ($x_4 = 0$) and has an odd number of nodes (an even number of unexplored nodes). *Agent*-1 (*Agent*-2) will be responsible for exploring the odd (even) nodes in the order $L(T)$.

We construct first the exploration sequence $\mathbb{Y}_T$ for *Agent*-2. Initially $\mathbb{Y}_T = \langle w'_1 = s, w'_2, \ldots, w'_{2p-1} = s \rangle$ is the depth-first traversal of the p internal nodes $w_1, w_2, \ldots, w_p$ of T. For each internal node w in T, if $v_{2(i+1)}, v_{2(i+2)}, \ldots, v_{2(i+k)}$ are the children of w which are at even positions in $L(T)$, then replace in $\mathbb{Y}_T$ the first occurrence of w with the sequence $w, v_{2(i+1)}, w, v_{2(i+2)}, w, \ldots, v_{2(i+k)}, w$. That is, *Agent*-2 traverses the internal nodes of the tree in the depth-first manner, and whenever it arrives during this traversal at an internal node w for the first time, before proceeding to the next node it first explores all children of w which are even nodes of T. The exploration sequence $\mathbb{X}_T$ for *Agent*-1 is constructed analogously. Since T has an odd number of nodes, both sequences $\mathbb{Y}_T$ and $\mathbb{X}_T$ have the same length $2p - 1 + (n - 1) = x_1 + 3x_3 + 1$. Lemma 6

says how these sequences are actually properly synchronized to form a valid (feasible) exploration scheme for T. It can be proven by induction, considering different kinds of relative positions of nodes v_{2i-2}, v_{2i-1}, v_{2i} and their parents. Lemma 7 follows from Lemma 6 and the formula for the length of sequences $\mathbb{Y}_T$ and $\mathbb{X}_T$.

Lemma 6. *Let T be a tree rooted at s with $2q+1 \geq 3$ nodes and with no* type-4 *nodes. For the exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ and $i = 1, \ldots, q$,*

1) there exist the i-th meeting step in $\mathcal{E}_T$; let $m(i)$ be that step;

2) the set of the explored nodes at step $m(i)$ is $S_{m(i)} = \{s\} \cup \{v_1, \ldots, v_{2i}\}$;

3) the meeting point at the i-th meeting step is the parent of node v_{2i}.

Lemma 7. *Let T be a tree rooted at s which has $n = 2q+1 \geq 3$ nodes and does not have any* type-4 *nodes. The exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ is valid, can be computed in linear time and its cost is equal to $x_1 + 3x_3$.*

Now we consider a general tree T, which may have *type-4* nodes. For each *type-4* node v in T, we add a new leaf l with parent v, placing l at the end of the list of the children of v. If the total number of nodes, including the added nodes, is even, then we add one more leaf as the last child of an arbitrary internal node. The obtained ordered tree T' is as required in Lemma 7. We obtain $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ for T from $\mathcal{E}_{T'} = (\mathbb{X}_{T'}, \mathbb{Y}_{T'})$ for T' by replacing the traversals of the added edges with waiting: if an added leaf l is, say, an odd node in T' and has parent v, then replace l in $\mathbb{X}_{T'}$ with v. Tree T' has $x'_1 = x_1 + x_4 + \beta$ leaves, for $\beta \in \{0, 1\}$, and $x'_3 = x_3 + x_4$ *type-3* nodes. Thus, using Lemma 7, the cost of $\mathcal{E}_T$ is as given in the following lemma.

Lemma 8. *The exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ for a rooted tree T is valid, can be computed in linear time and its cost is at most $x_1 + 3x_3 + 4x_4 + 1$.*

4.2 Generating a Good Spanning Tree of a Graph

We describe now our algorithm *Generate-Tree*(G, s) which computes a spanning tree T_G of a graph G such that the cost of the exploration scheme for G computed by algorithm *Search-Tree*(T_G, s) has cost at most 7/2 times worse than the minimum cost of an exploration scheme for G. Algorithm *Generate-Tree* tries to minimize the number of *type-4* nodes. Following the terminology used in [2], we define as BUSHY the rooted trees in which each internal node has at least two children. Bushy trees do not have *type-4* nodes. Algorithm *Search-Tree* computes for a bushy tree the same exploration scheme as the algorithm proposed in [2], and it was proven in [2] that that exploration scheme is optimal.

Algorithm *Generate-Tree* uses procedure *Bushy-Tree*(G', v) which for a given graph $G' = (V', E')$ and a node $v \in V'$ computes a maximal bushy tree T in G' rooted at v (that is, there is no bushy tree T' rooted at v such that $T \subsetneq T' \subseteq E'$).

Algorithm *Generate-Tree* consists of three steps. In Step 1 we compute vertex-disjoint trees $T_0, T_1, \ldots, T_k$, using procedure *Bushy-Tree*. Tree T_0 is rooted at s.

Tree T_i, $i = 1, 2 \ldots, k$ is returned by *Bushy-Tree*(G', v), where G' is the subgraph of G induced by the set of nodes not covered by the previous trees $T_0, \ldots, T_{i-1}$ and v is an arbitrary node in G' with degree (in G') at least 3. At the end of Step 1, the graph G' induced by the remaining uncovered nodes does not have a node of degree greater than 2. In Step 2 the nodes of G' (EXTERIOR NODES) are appended to the trees $T_0, T_1, \ldots, T_k$ by creating shortest paths to the leaves of these trees. In the final Step 3 all trees $T_0, T_1, \ldots, T_k$ are linked together into a spanning tree T_G of G. Figure 2 shows an example of a final spanning tree, including the details about the types of the nodes used later in the analysis. The algorithm can be implemented to run in linear time.

Algorithm 1 Algorithm Generate-Tree (G, s)

Step 1 (collecting bushy trees):
$T_0 \leftarrow$Bushy-Tree (G,s); $F \leftarrow \emptyset$;
Let G' be the subgraph of G induced by the nodes of G not in T_0;
while there exists a node u in $G' : d_{G'}(u) \geq 3$ **do**
  $T \leftarrow$Bushy-Tree (G',u); $F \leftarrow F \cup \{T\}$;
  Remove from G' the nodes in T and all edges incident to them;
end while
Let X be the set of nodes still in G' (*exterior nodes*);
Step 2 (appending exterior nodes):
Let X_e be the set of nodes in X adjacent to trees in $\{T_0\} \cup F$ (*type-e* nodes);
Append each node in X_e to one of the trees T_i adjacent to it;
Let $X_m = X \setminus X_e$ (*type-m* nodes);
Append nodes in X_m by creating shortest paths to *type-e* nodes;
Step 3 (linking the trees):
$T_G \leftarrow T_0$;
while $F \neq \emptyset$ **do**
  Find an edge (u, v) such that $u \in T_G$ and $v \in T \in F$;
  Add (u, v) and T to T_G; $F \leftarrow F \setminus \{T\}$;
end while

The set X of exterior nodes can then be partitioned into the nodes adjacent to a leaf of some T_i (*type-e* nodes) and the nodes not adjacent to any node of T_i (*type-m* nodes). Note that we use two classifications of the nodes of G. The first one partitions the nodes into three *types 1, 3* and *4*, and the second one which assigns *types e* and *m* to some of the nodes. We will need to introduce further types and sub-types to the second classification, We will need also to refer to intersections of types from these two classifications; for example, a *type-4e* node is a node which is both a *type-4* and a *type-e* node.

Lemma 9. *Each* type-m *node has degree at most 2 in G. For each maximal path $\langle v_0, v_1, \ldots, v_h \rangle$ of* type-m *nodes in G, one of the two end nodes is adjacent to a* type-e *node, while the other has degree 1 or is adjacent to a* type-e *node.*

On the basis of the above lemma, the computation done during Step 2 can be viewed in the following way. Append first each *type-e* node to a leaf of a tree

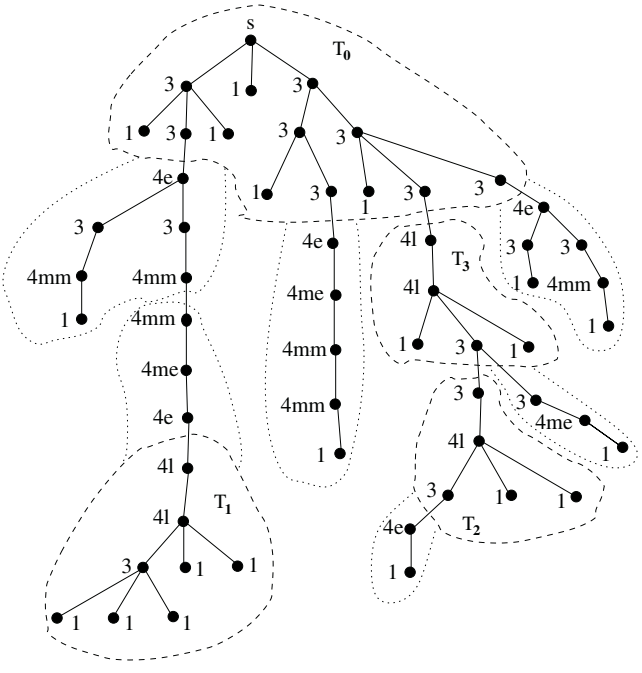


Fig. 2. An example of a final spanning tree computed by algorithm *Generate-Tree*. Types of nodes 1, 3 or 4, and further subdivision of *type-4* nodes are indicated

T_i, and then consider the maximal paths of *type-m* nodes. Append the length-0 paths and the paths with one end node having degree 1 (in G) to the adjacent *type-e* nodes. For each remaining path, remove its middle edge, breaking the possible tie arbitrarily, and append the resulting two paths to the adjacent *type-e* nodes (after the removal of the middle edges, each path is adjacent to exactly one *type-e* node). We sub-divide the *type-m* nodes into the *type-me* nodes which are adjacent to *type-e* nodes and the remaining *type-mm* nodes. During Step 2 paths composed of one *type-e* node, possibly one *type-me* node and possibly one or more *type-mm* nodes are appended to leaves of the trees T_i. During merging of tree in Step 3, at most 2 nodes of each appended tree may become *type-4* nodes. We call them *type-4l* nodes.

4.3 Approximation Ratio of the *STE* Algorithm

Lemma 8 implies that the cost of the exploration scheme computed by the *STE* algorithm for graph G and the starting node s is

$$t_{ALG} \leq x_1 + 3x_3 + 4x_4 + 1, \tag{1}$$

where x_i is the number of *type-i* nodes in tree T_G. Since the cost of the optimal exploration scheme is at least $t_{OPT} \geq n - 1 = x_1 + x_3 + x_4$, the number x_4 should be closely analysed. We provide first a bound on $x_{4e} + x_{4me} + x_{4l}$

(Lemma 10). We then use the number x_{4mm} to strengthen the lower bound on the cost of the optimal exploration scheme: no exploration scheme can keep exploring the *type-4mm* nodes at the average rate of one node per one step (Lemma 11). Lemmas 10 and 11 and the bound (1) imply our final result stated in Theorem 1.

Lemma 10. *For tree T_G computed by algorithm Generate-Tree(G, s),*

$$x_{4e} + x_{4me} + x_{4l} \leq 5x_1 + x_3 + x_{1mm} - 10. \tag{2}$$

Proof. Consider the maximal paths of *type-4* nodes in T_G, distinguishing two kinds of such paths. In an *leaf path* the last node (the node furthest from the root) has only one child (which must be a leaf); while in a *mid-tree path* the last node has at least two children. Each leaf path contains at most one *type-4e* node, at most one *type-4me* node and possibly one or more *type-4mm* nodes. Moreover, if a *type-4me* node is present in such a path, then the leaf attached to the last node of the path is a *type-1mm* node. Each mid-tree path contains at most two *type-4e* nodes, at most two *type-4me* nodes, at most two *type-4l* nodes and any number of *type-mm* nodes. Hence, denoting by z' and z'' the number of the leaf paths and the number of the mid-tree paths, respectively, we have $x_{4e} \leq z' + 2z''$, $x_{4me} \leq x_{1mm} + 2z''$, and $x_{4l} \leq 2z''$, so

$$x_{4e} + x_{4me} + x_{4l} \leq 6z'' + z' + x_{1mm}. \tag{3}$$

The last node of a mid-tree path must be a branching node in T_G, so $z'' \leq x_1 - 2$. On the other hand, since different maximal path of *type-4* nodes are attached in T_G to a different *type-3* nodes, we have $z'' \leq x_3 - z'$. Thus

$$6z'' \leq 5(x_1 - 2) + x_3 - z', \tag{4}$$

and Inequalities (3) and (4) give immediately (2). □

Lemma 11. *The minimum cost of an exploration scheme for an n-node graph G is*

$$t_{OPT} \geq n - 1 + \frac{1}{2}x_{mm} = x_1 + x_3 + x_4 + \frac{1}{2}(x_{1mm} + x_{4mm}). \tag{5}$$

Theorem 1. *For any graph G, the ratio of the cost t_{ALG} of the exploration scheme computed for G by the STE algorithm to the cost t_{OPT} of an optimal exploration scheme for G is at most 7/2.*

5 Limitations of BHS Based on Spanning Trees

The approximation algorithm for the BHS problem in arbitrary graphs which we presented in the previous section was based on the following two-part approach. Find first a suitable spanning tree T of the graph to explore, and then explore T

using a good BHS for trees. We show now that no graph exploration using this technique can guarantee a better approximation ratio than 3/2.

Let $G_c = (V, E)$ be an odd-length cycle with $V = \{v_1, v_2, \ldots, v_c\}$ and $E = \{(v_1, v_2), \ldots, (v_{c-1}, v_c), (v_c, v_1)\}$. A new graph G'_c is obtained from G_c using the construction for the NP-hardness proof given in Section 3, taking edge (v_c, v_1) as (x, y), with the following modification. The construction from Section 3 would add two shortcut edges for each node $v \in V \cup \{s\}$, but we add only one. If we trace the cycle $\langle s, v_1, v_2, \ldots, v_c, s\rangle$ in a planar embedding of G'_c, then the shortcut edges alternate between both sides of the cycle. Graph G'_7 is shown in Figure 3. Graph G'_c has $4c+3$ nodes and, using an argument as in the proof of Lemma 4, one can show that the cost of an optimal exploration scheme for G'_c is $4c+2$.

Consider the spanning tree of G'_c as shown in Figure 3. In the notation from Section 4.1, this tree has $x_3 = c-1$ *type-3* nodes $(v_1, v_2, \ldots, v_{c-1})$ and $x_1 = 3c+3$ *type-1* nodes. Lemma 7 implies that the cost of the exploration scheme computed for this tree by algorithm *Search-Tree* from Section 4.1 is $x_1 + 3x_3 = 6c$. We show now that this is essentially the best what an exploration scheme for a spanning tree of G'_c can do. Lemma 12, proven in [2], and Lemma 13 imply that the cost of any exploration scheme for any spanning tree of G'_c is at least $6c-2$, so at least $3/2 - O(1/c)$ times higher than the optimal cost of exploring G'_c.

Lemma 12. *[2] Let $T = (V_T, E_T, s)$ be a rooted tree with $n+1$ nodes. Let x_β and x_γ denote the number of nodes in $V_T \setminus \{s\}$ with exactly one descendant (*type-β *nodes) and with at least two descendants (*type-γ *nodes), respectively. Then the cost of any exploration scheme for T is at least $n + x_\beta + 2x_\gamma$.*

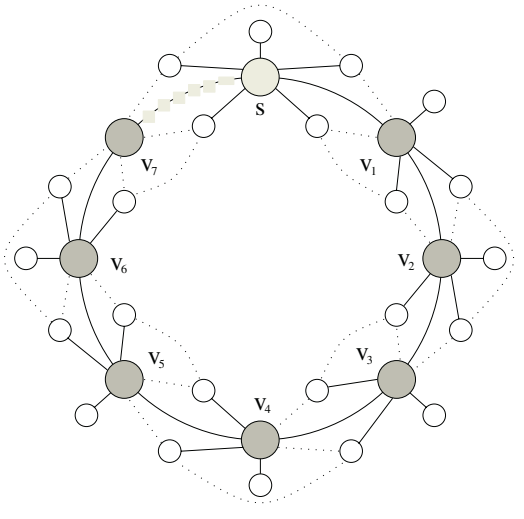


Fig. 3. Graph G'_7 and its "good" spanning tree (solid edges)

Lemma 13. *For any spanning tree T of G'_c rooted at s, $x_\beta + 2x_\gamma \geq 2c - 4$.*

Proof. Each node in $V \setminus \{v_c\}$ has at least one descendant. Let z be the number of *type-β* nodes in $V \setminus \{v_c\}$. At least $z - 2$ shortcut edges must belong to T: for each *type-β* node u in $V \setminus \{v_c\}$ except for at most two such nodes, the shortcut edge of u must be in T. Thus there are at least $z - 2$ twin nodes of type β or γ, so $x_\beta + 2x_\gamma \geq z + 2(c - 1 - z) + z - 2 = 2c - 4$. □

6 Conclusion

We proved that producing an optimal exploration scheme for an arbitrary graph is NP-hard, thus solving an open problem stated in [2]. We also gave a polynomial time 7/2-approximation algorithm for the BHS problem, which improves the ratio of 4 observed in [2]. Finally, we showed that any BHS that explores a graph via some spanning tree, as our algorithm does, cannot have an approximation ratio better than 3/2. A natural open problem is to decrease the 7/2 approximation ratio. It would also be interesting to generalize the model described in Section 2, and to investigate complexity issues, for the case of many agents and black holes.

References

1. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. 2004. manuscript.
2. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *Proc. of 8th International Conference on Principles of Distributed Systems (OPODIS 2004)*, pages 34–35, 2004.
3. S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Black hole search by mobile agents in hypercubes and related networks. In *Proc. of 6th International Conference on Principles of Distributed Systems (OPODIS 2002)*, pages 169–180, 2002.
4. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile agents searching for a black hole in an anonymous ring. In *Proc. of 15th International Symposium on Distributed Computing (DISC 2001)*, pages 166–179, 2001.
5. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. In *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 153–161, 2002.
6. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple agents rendezvous on a ring in spite of a black hole. In *Proc. 7th International Conference on Principles of Distributed Systems (OPODIS 2003)*, 2003.
7. M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.
8. F. Hohl. Time limited black box security: Protecting mobile agents from malicious hosts. In *Proc. Conf. on Mobile Agent Security*, LNCS 1419, pages 92–113, 1998.
9. F. Hohl. A framework to protect mobile agents by using reference states. In *Proc. 20th Int. Conf. on Distributed Computing Systems (ICDCS 2000)*, pages 410–417, 2000.

10. S. Ng and K. Cheung. Protecting mobile agents against malicious hosts by intention of spreading. In *Proc. Int. Conf. on Parallel and Distributed Processing and Applications (PDPTA'99)*, pages 725–729, 1999.
11. T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. Conf. on Mobile Agent Security*, LNCS 1419, pages 44–60, 1998.
12. K. Schelderup and J. Ines. Mobile agent security – issues and directions. In *Proc. 6th Int. Conf. on Intelligence and Services in Networks*, LNCS 1597, pages 155–167, 1999.
13. J. Vitek and G. Castagna. Mobile computations and hostile hosts. In D. Tsichritzis, editor, *Mobile Objects*, pages 241–261. University of Geneva, 1999.

On Semi-perfect 1-Factorizations*

Rastislav Královič and Richard Královič

Department of Computer Science,
Faculty of Mathematics, Physics and Informatics,
Comenius University, Bratislava, Slovakia

Abstract. The *perfect 1-factorization conjecture* by A. Kotzig [7] asserts the existence of a 1-factorization of a complete graph K_{2n} in which any two 1-factors induce a Hamiltonian cycle. This conjecture is one of the prominent open problems in graph theory. Apart from its theoretical significance it has a number of applications, particularly in designing topologies for wireless communication. Recently, a weaker version of this conjecture has been proposed in [1] for the case of *semi-perfect 1-factorizations*. A semi-perfect 1-factorization is a decomposition of a graph G into distinct 1-factors $F_1, \ldots, F_k$ such that $F_1 \cup F_i$ forms a Hamiltonian cycle for any $1 < i \leq k$. We show that complete graphs K_{2n}, hypercubes Q_{2n+1} and tori $T_{2n \times 2n}$ admit a semi-perfect 1-factorization.

1 Introduction

In this paper we deal with 1-factorizable graphs, i.e. graphs whose edges can be decomposed into 1-factors (perfect matchings). Clearly, taking the union of any two 1-factors F_i and F_j gives a 2-factor: a spanning subgraph consisting of a set of vertex-disjoint cycles. Additionally, if $F_i \cup F_j$ is connected it forms a Hamiltonian cycle and the corresponding 1-factors F_i, F_j are said to form a *perfect pair*.

It is widely known that a complete graph K_{2n} is 1-factorizable, see e.g. [8] for a survey. In his 1963 paper [7], A. Kotzig conjectured that for every $n \geq 2$ the complete graph K_{2n} can be decomposed into $n-1$ one-factors in such a way that any two of them form a perfect pair. Despite an extensive effort, this conjecture is still open. Currently it is known that such *perfect factorization* exists if either n is prime, $2n-1$ is prime or $2n \in \{16, 28, 36, 40, 50, 126, 170, 244, 344, 730, 1332, 1370, 1850, 2198, 3126, 6860\}$ (the references can be found in [10]).

One possible application of the perfect 1-factorization comes from the area of wireless communication. In [3] the problem of building a topology for an ad-hoc network of Bluetooth devices is addressed. As each device can communicate with exactly one other device at a time, the communication pattern at a given time forms a matching. In a bandwidth-efficient topology, a number k of 1-factors is used for communication in a time-multiplexed fashion, where k is

* Supported in part by grant APVT-20-018902.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 216–230, 2005.

the parameter of the network: larger values of k increase the robustness and decrease the diameter of the network while introducing more communication overhead due to interference. To achieve fairness, the chosen 1-factors should cover all edges. Moreover, any two of them should form a connected graph. In [3], a 1-factorization of K_{2n} into $F_1, \ldots, F_{n-1}$ is presented, such that any two 1-factors from $\{F_1, \ldots, F_p\}$ form a perfect pair, where p is the smallest prime factor of $2n-1$. Hence, the possible choices of the parameter k depend on the number of vertices which is not very convenient.

As a step towards the general solution, [1, 6] propose to solve a weaker version of the conjecture: find a 1-factorization of a graph G into 1-factors $F_1, \ldots, F_k$ such that $F_1 \cup F_i$ forms a Hamiltonian cycle for any $1 < i \leq k$. Such a 1-factorization will be called *semi-perfect*.

The semi-perfect 1-factorization conjecture has application also to the topological graph theory. In [5], the genus of joins and compositions of graphs is studied. According to [1]:

> *This kind of edge coloring* (i.e. semi-perfect 1-factorization) *of the cubes (and graphs in general) would lead to an improvement of existing genus results for joins and compositions of these graphs...*

In this paper we show that complete graphs K_{2n}, hypercubes Q_{2n+1} and tori $T_{2n\times 2n}$ admit a semi-perfect 1-factorization.

The potential of semi-perfect 1-factorizations in the topology design is yet to be investigated.

2 Preliminaries

Unless stated otherwise, we consider simple undirected graphs. A 1-factor (i.e. perfect matching) of a graph G is a spanning subgraph in which all vertices have degree 1. A 1-factorization is a decomposition of the set of edges into distinct 1-factors.

A 1-factorization of a graph G into 1-factors $F_1, \ldots, F_k$ is called *semi-perfect* if $F_1 \cup F_i$ forms a Hamiltonian cycle (i.e. connected 2-factor) for any $1 < i \leq k$.

Instead of constructing the particular 1-factors F_i, we shall construct the respective Hamiltonian cycles $F_1 \cup F_i$ according to the following definition:

Definition 1. *Let G be a graph, P be a 1-factor of G and $H_1, \ldots, H_k$ be Hamiltonian cycles of G such that each H_i contains P as its subset. Furthermore, let each edge of $G \setminus P$ be covered by exactly one cycle H_i. We call the set $\{H_1, \ldots, H_k\}$ a* P-cover of G.

It is easy to see that from given G, P and $H_1, \ldots, H_k$ that form a P-cover of G we can construct a semi-perfect 1-factorization of G. Indeed, let $P_1 := P$ and $P_{i+1} := H_i \setminus P$ for each $1 \leq i \leq k$. The 1-factors $P_1, \ldots, P_{k+1}$ form a semi-perfect 1-factorization of G.

In order to construct semi-perfect 1-factorizations of complete graphs and hypercubes we shall investigate the following two graph operations:

Definition 2. *Let $G = (V, E)$ be a graph. We define a graph $O_1(G)$ as follows: Each vertex v is replaced by vertices v_0 and v_1 connected by an edge. Each edge (u, v) is replaced by edges (u_0, v_0), (u_1, v_1), (u_0, v_1), (u_1, v_0). Formally,*

$$O_1(G) = (V', E'), \quad V' = \{v_0, v_1 \mid v \in G\}$$

$$E' = \{(v_0, v_1) \mid v \in G\} \cup \{(u_0, v_0), (u_1, v_1), (u_0, v_1), (u_1, v_0) \mid (u, v) \in E\}$$

Similarly, we define $O_2(G)$ as

$$O_2(G) = (V', E'), \quad V' = \{v_0, v_1 \mid v \in G\}$$

$$E' = \{(v_0, v_1) \mid v \in G\} \cup \{(u_0, v_0), (u_1, v_1) \mid (u, v) \in E\}$$

The graph $O_2(G)$ is the usual Cartesian product $G \times K_2$.

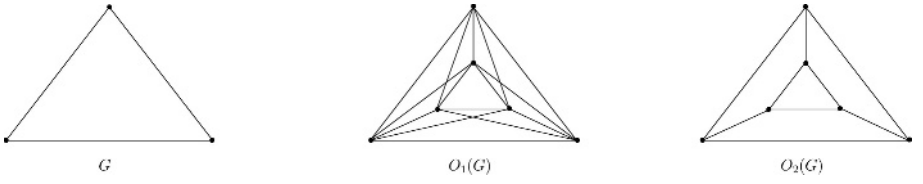


Fig. 1. The operations $O_1(G)$ and $O_2(G)$

3 Complete Graphs

In this section we show that complete graphs K_{2n} admit a semi-perfect 1-factorization. To do so we use the facts that $K_{2n} \cong O_1(K_n)$ and the directed complete graph K_n^* has a Hamiltonian decomposition, i.e. the arcs of K_n^* can be decomposed into disjoint Hamiltonian cycles.

Lemma 1. *Let G be an undirected graph without loops and multiple edges and G^* be a directed graph obtained from G by replacing each edge by two opposite arcs. Let $\mathcal{H} = \{H_1, \ldots, H_k\}$ be a set of Hamiltonian cycles on G^*, such that each arc of G^* is covered exactly once by $\mathcal{H}$. Let $G' = O_1(G)$ and $P = \{(v_0, v_1) \mid v \in V(G)\}$ be a 1-factor of G'. Then there exists a P-cover of G', i.e. a semi-perfect 1-factorization of G'.*

Proof. We shall create the P-cover $\mathcal{H}' = \{H'_1, \ldots, H'_{2k}\}$ of the graph G' by constructing two Hamiltonian cycles H'_{2i-1}, H'_{2i} of G from each Hamiltonian cycle H_i.

Without loss of generality let $V(G) = \{1, \ldots, n\}$. Let $H_i = \{u_1, \ldots, u_n\}$, where $u_1 = 1$. Every second edge of the cycles H'_{2i-1} and H'_{2i} will be from P, i.e. of the form $\{x_0, x_1\}$ for some x in $V(G)$. We construct a Hamiltonian cycle H'_{2i-1} as a sequence of vertices $(v_1, \ldots v_{2n})$, such that for the $2j-1$-st edge of the sequence, denoted e_{2j-1}, it holds $e_{2j-1} = \{v_{2j-1}, v_{2j}\} = \{x_0, x_1\}$ where $x = u_j$.

What remains to be set is the orientation of the edge e_{2j-1} in the sequence, i.e. whether $v_{2j-1} = x_0$ and $v_{2j} = x_1$ or vice versa.

We say that the edge e_{2j-1} is oriented positively if $v_{2j-1} = x_0 \wedge v_{2j} = x_1$ and negatively if $v_{2j-1} = x_1 \wedge v_{2j} = x_0$. Define the orientation of e_{2j-1} inductively in the following way: Edge e_1 is oriented positively, i.e. $v_1 = u_{10} = 1_0$ and $v_2 = u_{11} = 1_1$. If $u_{j-1} < u_j$ then edge e_{2j-1} has the same orientation as the edge e_{2j-3}, otherwise it has opposite orientation.

The Hamiltonian cycle H'_{2i} is constructed in the same way as H'_{2i-1}, except that orientation of edges e_{2j-1} for all $j > 1$ is flipped.

From the construction of $\mathcal{H}'$ it follows that each H'_i contains the matching P as its subset. It remains to show that each edge of $E(G') \setminus P$ is covered by exactly one H'_i.

Let $\{u, v\}$ be any edge of G such that $u < v$. Let H_x be the only member of $\mathcal{H}$ that covers the edge (u, v) and H_y be the only member of $\mathcal{H}$ that covers the edge (v, u). This situation is illustrated on Figure 2.

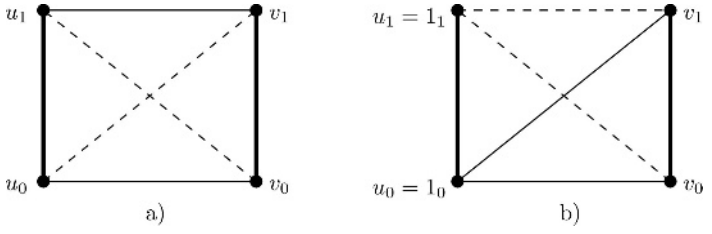


Fig. 2. Construction used in Lemma 1 where $u < v$, edge (u, v) is covered by H_x and edge (v, u) is covered by H_y. Dashed lines belong to cycles H'_{2x-1} and H'_{2x}. Thin solid lines belong to cycles H'_{2y-1} and H'_{2y}. The case $u > 1$ is presented on the left-hand side. The case $u = 1$ is presented on the right-hand side

Assume that $u > 1$. Obviously $x \neq y$, hence H_x and H_y contribute to four different Hamiltonian cycles on G': H'_{2x-1}, H'_{2x}, H'_{2y-1}, and H'_{2y}. The construction of $\mathcal{H}'$ ensures that edges $\{u_0, u_1\}$ and $\{v_0, v_1\}$ have the same orientation in H'_{2x-1} and H'_{2x} and opposite orientation in H'_{2y-1} and H'_{2y}. Thus it holds that edges $\{u_1, v_0\}$ and $\{u_0, v_1\}$ are covered only by H'_{2x-1} and H'_{2x} and edges $\{u_0, v_0\}$ and $\{u_1, v_1\}$ are covered only by H'_{2y-1} and H'_{2y}.

Now consider the case $u = 1$. Clearly the edges $\{1_1, v_0\}$, $\{1_1, v_1\}$ are covered only by the cycles H'_{2x-1} and H'_{2x}. Similarly, the edges $\{1_0, v_0\}$, $\{1_0, v_1\}$ are covered only by the cycles H'_{2y-1} and H'_{2y}. □

Now we plug the facts about complete graphs into the preceding lemma:

Theorem 1. *Let $n > 1$ and K_{2n} be a complete graph with $2n$ vertices. Then there exists a semi-perfect 1-factorization of K_{2n}.*

Proof. Obviously, the graph $K_{2n} \cong O_1(K_n)$. Assume that $n \notin \{4, 6\}$. By the result of Tillson [11] it is known that an oriented complete graph K_n^* is decomposable into $n-1$ Hamiltonian cycles for all $n \notin \{4, 6\}$. Hence according to Lemma 1 there exists a semi-perfect 1-factorization of K_{2n}.

For cases $n = 4$ and $n = 6$ we use similar approach as in Lemma 1: take the appropriate $n-1$ Hamiltonian cycles $\mathcal{H}$ on K_n and construct $2n-2$ Hamiltonian cycles on $O_1(K_n)$ that form a P-cover. Since $\mathcal{H}$ is not a decomposition of K_n^*, the technique for assigning orientations to e_j from Lemma 1 does not work. However, it is possible to find a suitable assignment of orientations.

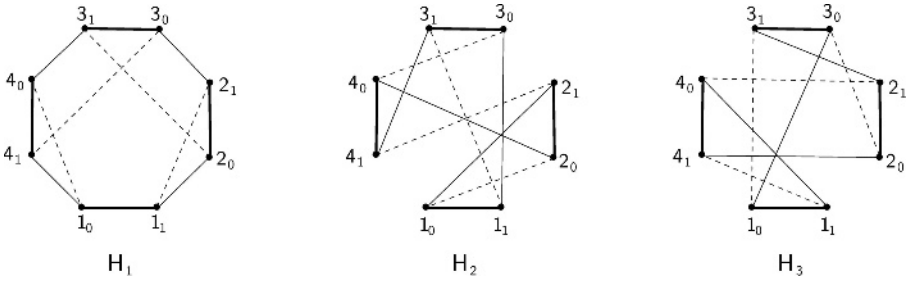


Fig. 3. A semi-perfect 1-factorization of $O_1(K_4) \cong K_8$

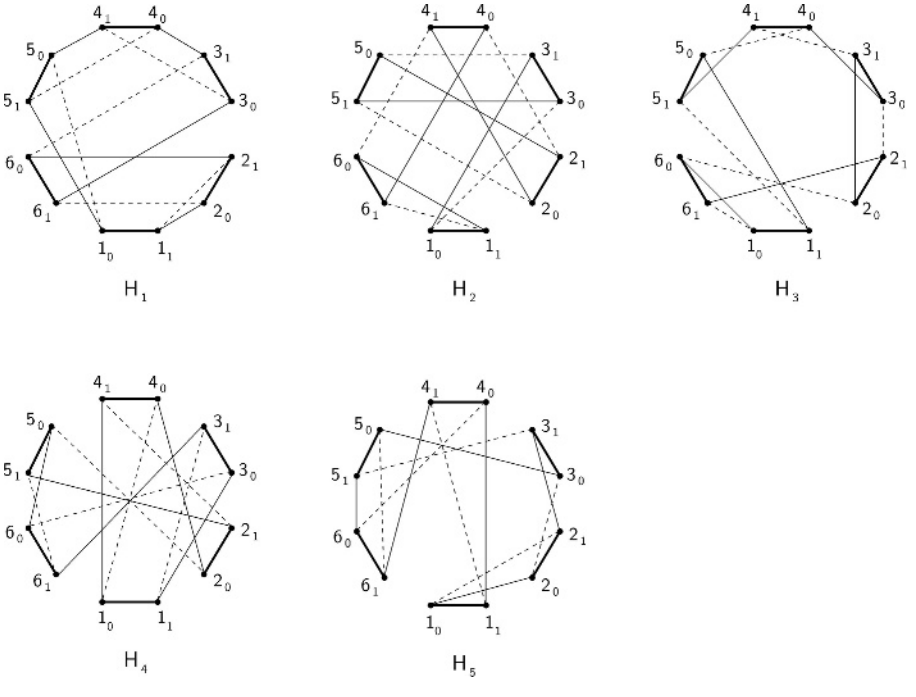


Fig. 4. A semi-perfect 1-factorization of $O_1(K_6) \cong K_{12}$

For $n = 4$ we take Hamiltonian cycles $H_1 = (1, 2, 3, 4)$, $H_2 = (1, 3, 4, 2)$ and $H_3 = (1, 4, 2, 3)$. The resulting semi-perfect 1-factorization of K_8 is shown on Figure 3.

For $n = 6$ we take Hamiltonian cycles $H_1 = (1, 2, 6, 3, 4, 5)$, $H_2 = (1, 6, 4, 2, 5, 3)$, $H_3 = (1, 5, 4, 3, 2, 6)$, $H_4 = (1, 3, 6, 5, 2, 4)$ and $H_5 = (1, 4, 6, 5, 3, 2)$. The resulting semi-perfect 1-factorization of K_{12} is shown on Figure 4. □

4 Hypercubes

To show that hypercubes Q_{2n+1} admit a semi-perfect 1-factorization we use the same technique as for complete graphs. We use the fact that $Q_n \cong O_2(Q_{n-1})$ and that an even hypercube is Hamiltonian decomposable.

Lemma 2. *Let G be an undirected graph without loops and multiple edges with an even number of vertices. Let $\mathcal{H} = \{H_1, \ldots, H_k\}$ be a set of Hamiltonian cycles on G, such that each edge of G is covered exactly once by $\mathcal{H}$. Let $G' = O_2(G)$ and $P = \{(v_0, v_1) \mid v \in V(G)\}$ be a matching of G'. Then there exists a P-cover of G', hence there exists a semi-perfect 1-factorization of G'.*

Proof. We shall create the P-cover $\mathcal{H}' = \{H'_1, \ldots, H'_{2k}\}$ of the graph G' by constructing two Hamiltonian cycles H'_{2i-1}, H'_{2i} on G from a Hamiltonian cycle H_i in a similar fashion as in Lemma 1. However, since edges $\{u_0, v_1\}$ and $\{u_1, v_0\}$ do not exist in G', the orientation of edges in the constructed cycles must be alternating. To ensure the correctness of the construction, the number of vertices of G must be even.

Let $n = 2m$, $V(G) = \{1, \ldots, 2m\}$ and $H_i = (u_1, \ldots, u_{2m})$, where $u_1 = 1$. We construct a Hamiltonian cycle H'_{2i-1} as a sequence of vertices $(v_1, \ldots, v_{4m})$, such that $v_{4j-3} = u_{2j-1_0}$, $v_{4j-2} = u_{2j-1_1}$, $v_{4j-1} = u_{2j_1}$ and $v_{4j} = u_{2j_0}$.

The Hamiltonian cycle H'_{2i} is constructed in the same way as H'_{2i-1}, except that the orientation of all edges is flipped: $v_{4j-3} = u_{2j-1_1}$, $v_{4j-2} = u_{2j-1_0}$, $v_{4j-1} = u_{2j_0}$ and $v_{4j} = u_{2j_1}$. This construction is illustrated by Figure 5.

Obviously each H'_i contains the matching P as its subset. Any edge of $E(G') \setminus P$ can be expressed as $\{u_q, v_q\}$ for some $q \in \{0, 1\}$. Let H_i be the only

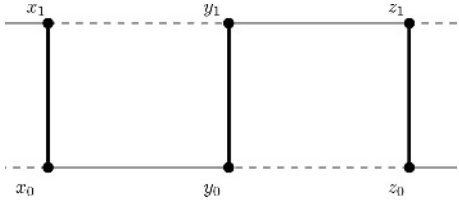


Fig. 5. Construction used in Lemma 2 where $x = u_{2j-1}$, $y = u_{2j}$ and $z = u_{2j+1}$ for some $H_i = (u_1, \ldots, u_{2m})$. Dashed lines belong to the cycle H'_{2i-1}. Thin solid lines belong to the cycle H'_{2i}

member of $\mathcal{H}$ that covers the edge $\{u, v\}$. The construction of $\mathcal{H}'$ ensures that the edge $\{u_q, v_q\}$ is covered by either H'_{2i-1} or H'_{2i}. Hence $\mathcal{H}'$ is a P-cover of the graph G'. □

Theorem 2. *Let $n \geq 1$ and Q_{2n+1} be a hypercube with 2^{2n+1} vertices. Then there exists a semi-perfect 1-factorization of Q_{2n+1}.*

Proof. It is easy to see that the hypercube Q_{2n+1} is isomorphic to $O_2(Q_{2n})$. By the result of Aubert and B. Schneider [2] (see also [9, 4]) it is known that a hypercube Q_{2n} is decomposable into n Hamiltonian cycles for all n. Hence according to Lemma 2 there exists a semi-perfect 1-factorization of Q_{2n+1}. □

5 Tori

In this section we show that a torus T_{2n} of size $2n \times 2n$ admits a semi-perfect factorization. Obviously, torus T_n where n is odd contains an odd number of vertices and thus does not admit a semi-perfect 1-factorization.

We shall denote the vertices of a torus T_n by pairs $[i, j]$, $i, j \in \{1, \ldots, n\}$, such that two vertices $[i, j]$ and $[k, l]$ are connected by an edge if and only if exactly one of the two conditions holds: $j = l \wedge |k-i| \in \{1, n-1\}$ or $k = i \wedge |j-l| \in \{1, n-1\}$. Edges with $|k - i| = n - 1$ or $|j - l| = n - 1$ are called wrap-around edges.

Since tori are 4-regular, a 1-factorization can be viewed as an edge coloring using 4 colors. This 1-factorization is semi-perfect if and only if edges of color 0 together with edges of any other color form a Hamiltonian cycle.

It is possible to embed a torus T_n in an infinite grid by cutting the wrap-around edges. Edges $\{[i, 1], [i, n]\}$ are replaced by edges $\{[i, 0], [i, 1]\}$ and $\{[i, n], [i, n + 1]\}$; edges $\{[1, i], [n, n]\}$ are replaced by edges $\{[0, i], [1, i]\}$ and $\{[n, i], [n + 1, i]\}$. By this operation we obtain an *embedded torus ET_n*.

Definition 3. *An* embedded torus *ET_n is a subgraph of infinite two-dimensional grid induced by vertices $\{1, \ldots, n\} \times \{1, \ldots, n\}$ united with the set of* external edges *E_{ext} defined as follows:*

$$E_{ext} = \big\{\{[0, i], [1, i]\},\ \{[n, i], [n+1, i]\},\ \{[i, 0], [i, 1]\},\ \{[i, n], [i, n+1]\} \mid 1 \leq i \leq n\big\}$$

Edges of ET_n that are not external are called internal edges.

A Hamiltonian cycle in a torus may form several paths in the corresponding embedded torus. All these paths are terminated by an external edge. We call these paths *partial paths*.

Definition 4. *Let ET be an embedded torus with a given edge 4-coloring. A path containing only edges of color 0 and color i for some fixed i is called a* partial path *if and only if it starts and ends with an external edge.*

We show that it is possible to color the edges of an embedded torus by 4 colors such that certain invariant about the partial paths is preserved.

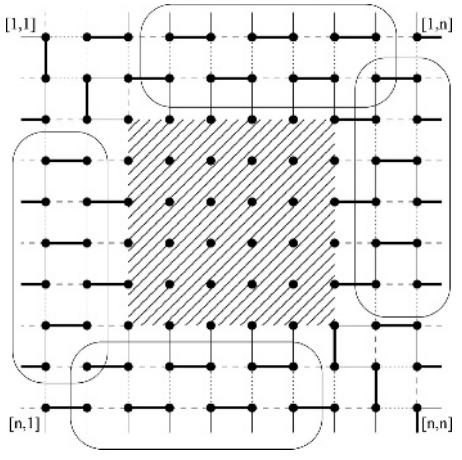


Fig. 6. Edge 4-coloring of an embedded torus used in Lemma 3. Color 0 is printed as bold lines, color 1 as thin lines, color 2 as dotted lines, and color 3 as dashed lines. Regularly repeating patterns are marked by ellipses. Formal description of this coloring is presented in the Appendix

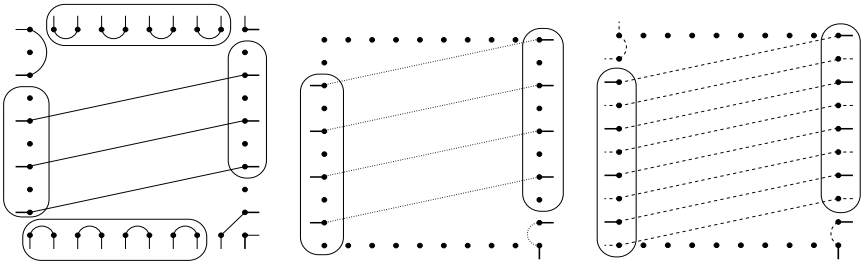


Fig. 7. Connectivity of partial paths used in Lemma 3. Left picture shows partial paths colored by 0 and 1, middle picture partial paths colored by 0 and 2, and right picture paths colored by 0 and 3. Regularly repeating patterns are marked by ellipses. Formal description is presented in the Appendix

Lemma 3. *Let ET_n be an embedded torus of size $n \times n$, such that $n \geq 4$ is even. Then it is possible to color its edges in such a way that the external edges are colored as in Figure 6. Moreover, each edge is covered by some partial path and the external edges are connected by partial paths according to Figure 7. Exact formulation of these invariants is presented in the Appendix.*

Proof. We shall construct the described coloring of ET_n inductively. The appropriate coloring for $n = 4$ and $n = 6$ is presented on Figure 8.

Now assume that $n \geq 8$ and we already have an appropriate coloring for an embedded torus ET' of size $n - 4 \times n - 4$. We insert the torus ET' into the embedded torus ET of size $n \times n$ such that a vertex $[i, j]$ is mapped onto the vertex $[i + 2, j + 2]$, hence vertices of the torus ET' are mapped onto vertices

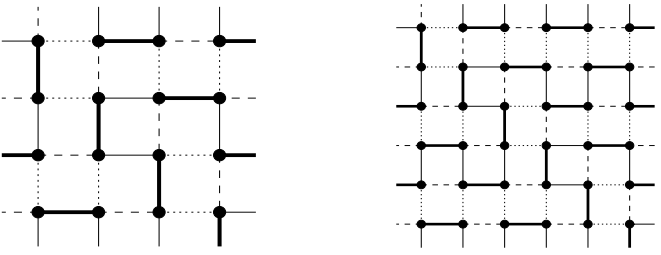


Fig. 8. Colorings of embedded tori ET_4 and ET_6 used in Lemma 3. Color 0 is printed as bold lines, color 1 as thin lines, color 2 as dotted lines and color 3 as dashed lines

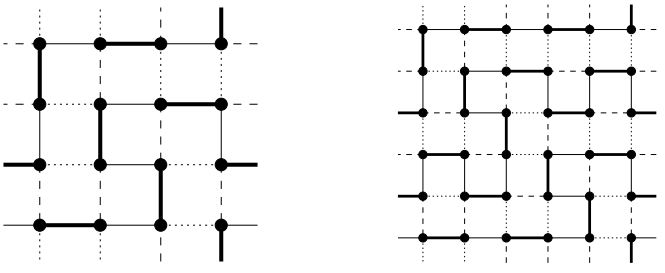


Fig. 9. Colorings of the tori T_4 and T_6 that induce a semi-perfect 1-factorization

$\{3, \ldots, n-2\} \times \{3, \ldots, n-2\}$. Edges not covered by ET' (i.e. edges incident to $V(ET) \backslash V(ET')$) are colored according to Figure 6. It is straightforward to check that this coloring is consistent with the external edges of ET' and preserves the invariant about the connectivity of the partial paths presented on Figure 7. Also, it is easy to see that each edge of ET is covered by some partial path. □

Theorem 3. *Let T_n be a torus of size $n \times n$ such that $n \geq 4$ is even. Then T_n admits a semi-perfect 1-factorization.*

Proof. We find an edge 4-coloring of the torus T_n that induces a semi-perfect 1-factorization. In case $n = 4$ or $n = 6$ the appropriate 4-coloring is shown on Figure 9.

Assume that $n \geq 8$. We obtain a coloring of a torus T of size $n \times n$ similarly as in Lemma 3. Let ET' be an embedded torus of size $n-4 \times n-4$ colored according to Lemma 3. We insert a torus ET' into T such that a vertex $[i, j]$ is mapped onto the vertex $[i+2, j+2]$ and color the remaining edges according to the Figure 10. It is easy to see that this coloring is consistent with the coloring of the external edges of ET'.

It remains to show that invariants about partial paths granted by Lemma 3 and the coloring presented on Figure 10 ensure that the edges of color 0 together with the edges of any other color form a Hamiltonian cycle of T_n.

Analogically to Lemma 3, it is easy to see that all non-wrap-around edges of T are covered by some partial path of the embedded torus corresponding to the torus T. Hence, to prove that edges of colors 0 and $c \in \{1, 2, 3\}$ form a

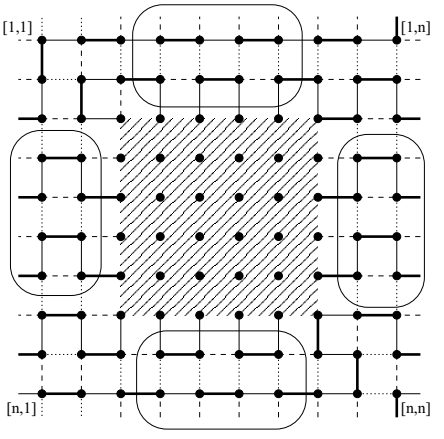


Fig. 10. Edge 4-coloring of a torus used in Theorem 3. Color 0 is printed as bold lines, color 1 as thin lines, color 2 as dotted lines and color 3 as dashed lines. Regularly repeating patterns are marked by ellipses. Formal description of this coloring is presented in the Appendix

Hamiltonian cycle, it is sufficient to show that all wrap-around edges of colors 0 and c are covered by one cycle consisting of edges of colors 0 and c.

For $c = 1$ it is easy to verify this fact. Indeed, the wrap-around edges of colors 0 and 1 occur on a cycle in the following order: $\{[n,n],[n,1]\},\{[n-1,n],[n-1,1]\},\{[n-3,n],[n-3,1]\},\ldots,\{[n-2i-1,n],[n-2i-1,1]\},\ldots,\{[3,n],[3,1]\},\{[1,n],[n,n]\}$

Similar statement holds for $c = 2$. The order of the wrap-around edges is: $\{[1,1],[n,1]\},\{[n,2],[1,2]\},\{[1,n],[n,n]\},\{[n-1,n],[n-1,1]\},\{[n-3,n],[n-3,1]\},\ldots,\{[n-2i-1,n],[n-2i-1,1]\},\ldots,\{[3,n],[3,1]\}$

And the order of wrap-around edges for $c = 3$ is: $\{[n-1,n],[n-1,1]\},\{[n-3,n],[n-3,1]\},\ldots,\{[n-2i-1,n],[n-2i-1,1]\},\ldots,\{[3,n],[3,1]\},\{[1,3],[n,3]\},\{[1,4],[n,4]\},\ldots,\{[1,i],[n,i]\},\ldots,\{[1,n-1],[n,n-1]\},\{[n-2,n],[n-2,1]\},\{[n-4,n],[n-4,1]\},\ldots,\{[n-2i,n],[n-2i,1]\},\ldots,\{[2,n],[2,1]\},\{[1,1],[1,n]\},\{[1,n],[n,n]\}$

Hence, we have proven that for every color $c \in \{1,2,3\}$ the edges of colors 0 or c form a Hamiltonian cycle of T_n. □

6 Conclusions and Further Research

We have shown that complete graphs K_{2n}, hypercubes Q_{2n+1} and tori $T_{2n\times 2n}$ admit a semi-perfect 1-factorization. The case of Q_{2n} remains open. As it seems, there is no direct relation between perfect and semi-perfect 1-factorizations, however it might be interesting to find another topologies admitting semi-perfect 1-factorization. Also, the potential of semi-perfect 1-factorizations in the topology design could be investigated.

References

1. Dan Archdeacon: Problems in Topological Graph Theory. http://www.emba.uvm.edu/ ~archdeac/problems/perfectq.htm
2. J. Aubert, B. Schneider: *Decompositions de la somme cartsienne d'un cycle et l'union de deux cycles hamiltoniens en cycles hamiltoniens*, Discrete Mathematics, 38(1982)7-16.
3. S. Baatz et al.: *Building Efficient Bluetooth Scatternet Topologies from 1-Factors*, Proc. IASTED International Conference on Wireless and Optical Communications, WOC 2002, pp. 300–305, Banff, Alberta, Canada, July 2002
4. Douglas W. Bass, I. Hal Sudborough: *Hamilton Decompositions and* $(n/2)$*-Factorizations of Hypercubes*, Journal of Graph Algorithms and Applications, 7(2003)79–98
5. D. Craft: *On the genus of joins and compositions of graphs*, Discrete Mathematics, 178(1998)25–50.
6. D. Craft, private communication
7. A. Kotzig: Hamilton graphs and Hamilton circuits Theory of Graphs and its Applications Proceedings of the Symposium of Smolenice, 1963 Nakl. ČSAV Praha, 1964, pp. 62–82
8. E. Mendelsohn, A. Rosa: One-Factorizations of the Complete Graph – a Survey Journal of Graph Theory, 9, 1985, 43–65
9. K. Okuda, S. W. Song: *Revisiting Hamiltonian Decomposition of the Hypercube*, Proc. XIII Symposium on Integrated Circuits and System Design, SBCCI2000, pp. 55–60, Manaus, Brazil, September 2000.
10. D.A. Pike: Hamiltonian decompositions of line graphs of perfectly 1-factorizable graphs of even degree Australian Journal of Combinatorics 12 (1995), 291–294
11. T.Tillson: A Hamiltonian decomposition of K_{2n}, $2n \geq 8$, J. Combin. Theory Ser. B 29 (1980) 68–74.

Appendix – Formal Formulation of Invariants

Formal version of Lemma 3

Let ET_n be an embedded torus of size $n \times n$, such that $n \geq 4$ is even. Let $c(e)$ be the color of the edge $e \in E(ET_n)$ and $m_c(e)$ be the last edge of the $(0, c)$-colored partial path beginning with the external edge $e \in E(ET_n)$. Then it is possible to color edges of ET_n such that all edges are covered by some partial path and the following conditions hold:

Top edges:

$$
\begin{aligned}
&c([0,1],[1,1]) &&= 3 && \\
&c([0,i],[1,i]) &&= 1 && \forall 2 \leq i \leq n
\end{aligned}
$$

Left edges:

$$
\begin{aligned}
&c([1,0],[1,1]) &&= 1 && \\
&c([2i,0],[2i,1]) &&= 3 && \forall 1 \leq i \leq \tfrac{n}{2} \\
&c([2i+1,0],[2i+1,1]) &&= 0 && \forall 1 \leq i \leq \tfrac{n}{2} - 1
\end{aligned}
$$

Bottom edges:

$c([n,i],[n+1,i]) = 1 \quad \forall 1 \le i \le n-1$
$c([n,n],[n+1,n]) = 0$

Right edges:

$c([2i+1,n],[2i+1,n+1]) = 0 \quad \forall 0 \le i \le \frac{n}{2}-1$
$c([2i,n],[2i,n+1]) = 3 \quad \forall 1 \le i \le \frac{n}{2}-1$
$c([n,n],[n,n+1]) = 1$

Partial paths:

$m_1([1,0],[1,1]) = \{[3,0],[3,1]\}$
$m_1([2i+1,0],[2i+1,1]) = \{[2i-1,n],[2i-1,n+1]\} \quad \forall 2 \le i \le \frac{n}{2}-1$
$m_1([n,2i+1],[n+1,2i+1]) = \{[n,2i+2],[n+1,2i+2]\} \quad \forall 0 \le i \le \frac{n}{2}-2$
$m_1([n,n-1],[n+1,n-1]) = \{[n-1,n],[n-1,n+1]\}$
$m_1([n,n],[n+1,n]) = \{[n,n],[n,n+1]\}$
$m_1([0,n],[1,n]) = \{[1,n],[1,n+1]\}$
$m_1([0,2i],[1,2i]) = \{[0,2i+1],[1,2i+1]\} \quad \forall 1 \le i \le \frac{n}{2}-1$
$m_2([2i+1,0],[2i+1,1]) = \{[2i-1,n],[2i-1,n+1]\} \quad \forall 1 \le i \le \frac{n}{2}-1$
$m_c([n,n],[n+1,n]) = \{[n-1,n],[n-1,n+1]\} \quad \forall c \in \{2,3\}$
$m_3([0,1],[1,1]) = \{[2,0],[2,1]\}$
$m_3([i,0],[i,1]) = \{[i-2,n],[i-2,n+1]\} \quad \forall 3 \le i \le n$

Proof. The proof is equivalent to the proof written in Section 5. The following coloring of the internal edges of ET not covered by ET' is used:

Top edges:

$c([1,1],[1,2]) = 2$
$c([1,2i],[1,2i+1]) = 0 \quad \forall 1 \le i \le \frac{n}{2}-1$
$c([1,2i+1],[1,2i+2]) = 3 \quad \forall 1 \le i \le \frac{n}{2}-1$
$c([1,1],[2,1]) = 0$
$c([1,2],[2,2]) = 3$
$c([1,i],[2,i]) = 2 \quad \forall 3 \le i \le n$
$c([2,1],[2,2]) = 2$
$c([2,2],[2,3]) = 1$
$c([2,2i+1],[2,2i+2]) = 0 \quad \forall 1 \le i \le \frac{n}{2}-1$
$c([2,2i],[2,2i+1]) = 3 \quad \forall 2 \le i \le \frac{n}{2}-1$

Bottom edges:

$c([n,2i+1],[n,2i+2]) = 0 \quad \forall 0 \le i \le \frac{n}{2}-2$
$c([n,2i],[n,2i+1]) = 3 \quad \forall 1 \le i \le \frac{n}{2}-1$
$c([n,n-1],[n,n]) = 2$
$c([n-1,i],[n,i]) = 2 \quad \forall 1 \le i \le n-2$
$c([n-1,n-1],[n,n-1]) = 0$
$c([n-1,n],[n,n]) = 3$
$c([n-1,2i+1],[n-1,2i+2]) = 3 \quad \forall 0 \le i \le \frac{n}{2}-2$
$c([n-1,2i],[n-1,2i+1]) = 0 \quad \forall 1 \le i \le \frac{n}{2}-2$
$c([n-1,n-2],[n-1,n-1]) = 1$
$c([n-1,n-1],[n-1,n]) = 2$

Left edges:

$c([2,1],[3,1]) = 1$
$c([2,2],[3,2]) = 0$
$c([2i+1,1],[2i+1,2]) = 3 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([2i,1],[2i,2]) = 0 \quad \forall 2 \leq i \leq \frac{n}{2}-1$
$c([2i+1,1],[2i+2,1]) = 2 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([2i+1,2],[2i+2,2]) = 2 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([2i,1],[2i+1,1]) = 1 \quad \forall 2 \leq i \leq \frac{n}{2}-1$
$c([2i,2],[2i+1,2]) = 1 \quad \forall 2 \leq i \leq \frac{n}{2}-1$

Right edges:

$c([n-2,n-1],[n-1,n-1]) = 3$
$c([n-2,n],[n-1,n]) = 1$
$c([2i+1,n-1],[2i+1,n]) = 3 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([2i,n-1],[2i,n]) = 0 \quad \forall 2 \leq i \leq \frac{n}{2}-1$
$c([2i+1,n-1],[2i+2,n-1]) = 2 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([2i+1,n],[2i+2,n]) = 2 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([2i,n-1],[2i+1,n-1]) = 1 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([2i,n],[2i+1,n]) = 1 \quad \forall 1 \leq i \leq \frac{n}{2}-2$

Formal Description of Figure 10

Let T be a torus of size $n \times n$ and ET' be an embedded torus of size $(n-4)\times(n-4)$ as in the proof of Theorem 3. Let $c(e)$ be the color of the edge $e \in E(T)$. The coloring of T used in the proof of Theorem 3 presented on the Figure 10 satisfies the following conditions:

Wrap-around edges

$c([1,1],[n,1]) = 2$
$c([1,2],[n,2]) = 2$
$c([1,i],[n,i]) = 3 \quad \forall 3 \leq i \leq n-1$
$c([1,n],[n,n]) = 0$
$c([1,1],[1,n]) = 3$
$c([2i,1],[2i,n]) = 3 \quad \forall 1 \leq i \leq \frac{n}{2}-1$
$c([2i+1,1],[2i+1,n]) = 0 \quad \forall 1 \leq i \leq \frac{n}{2}-1$
$c([n,1],[n,n]) = 1$

Edges on the outermost perimeter

Top edges:

$c([1,2i+1],[1,2i+2]) = 1 \quad \forall 0 \leq i \leq \frac{n}{2}-1$
$c([1,2i],[1,2i+1]) = 0 \quad \forall 1 \leq i \leq \frac{n}{2}-1$

Left edges:

$c([1,1],[2,1]) = 0$
$c([2i,1],[2i+1,1]) = 1 \quad \forall 1 \leq i \leq \frac{n}{2}-1$
$c([2i+1,1],[2i+2,1]) = 2 \quad \forall 1 \leq i \leq \frac{n}{2}-2$
$c([n-1,1],[n,1]) = 3$

Bottom edges:

$$\begin{aligned} c([n,2i+1],[n,2i+2]) &= 0 \quad \forall 0 \le i \le \tfrac{n}{2}-2 \\ c([n,2i],[n,2i+1]) &= 1 \quad \forall 1 \le i \le \tfrac{n}{2}-1 \\ c([n,n-1],[n,n]) &= 2 \end{aligned}$$

Right edges:

$$\begin{aligned} c([2i+1,n],[2i+2,n]) &= 2 \quad \forall 0 \le i \le \tfrac{n}{2}-2 \\ c([2i,n],[2i+1,n]) &= 1 \quad \forall 1 \le i \le \tfrac{n}{2}-1 \\ c([n-1,n],[n,n]) &= 3 \end{aligned}$$

Edges between the outermost and the 2nd outermost perimeter

Top edges:

$$\begin{aligned} c([1,2],[2,2]) &= 3 \\ c([1,i],[2,i]) &= 2 \quad \forall 3 \le i \le n-1 \end{aligned}$$

Left edges:

$$\begin{aligned} c([2,1],[2,2]) &= 2 \\ c([2i+1,1],[2i+1,2]) &= 3 \quad \forall 1 \le i \le \tfrac{n}{2}-2 \\ c([2i,1],[2i,2]) &= 0 \quad \forall 2 \le i \le \tfrac{n}{2}-1 \\ c([n-1,1],[n-1,2]) &= 2 \end{aligned}$$

Bottom edges:

$$\begin{aligned} c([n-1,2],[n,2]) &= 3 \\ c([n-1,i],[n,i]) &= 2 \quad \forall 3 \le i \le n-2 \\ c([n-1,n-1],[n,n-1]) &= 0 \end{aligned}$$

Right edges:

$$\begin{aligned} c([2i,n-1],[2i,n]) &= 0 \quad \forall 1 \le i \le \tfrac{n}{2}-1 \\ c([2i+1,n-1],[2i+1,n]) &= 3 \quad \forall 1 \le i \le \tfrac{n}{2}-2 \\ c([n-1,n-1],[n-1,n]) &= 2 \end{aligned}$$

Edges on the 2nd outermost perimeter

Top edges:

$$\begin{aligned} c([2,2],[2,3]) &= 1 \\ c([2,2i+1],[2,2i+2]) &= 0 \quad \forall 1 \le i \le \tfrac{n}{2}-2 \\ c([2,2i],[2,2i+1]) &= 3 \quad \forall 2 \le i \le \tfrac{n}{2}-1 \end{aligned}$$

Left edges:

$$\begin{aligned} c([2,2],[3,2]) &= 0 \\ c([2i+1,2],[2i+2,2]) &= 2 \quad \forall 1 \le i \le \tfrac{n}{2}-2 \\ c([2i,2],[2i+1,2]) &= 1 \quad \forall 2 \le i \le \tfrac{n}{2}-1 \end{aligned}$$

Bottom edges:

$$\begin{aligned}
c([n-1,2i],[n-1,2i+1]) &= 0 \quad \forall 1 \leq i \leq \tfrac{n}{2}-2\\
c([n-1,2i+1],[n-1,2i+2]) &= 3 \quad \forall 1 \leq i \leq \tfrac{n}{2}-2\\
c([n-1,n-2],[n-1,n-1]) &= 1
\end{aligned}$$

Right edges:

$$\begin{aligned}
c([2i,n-1],[2i+1,n-1]) &= 1 \quad \forall 1 \leq i \leq \tfrac{n}{2}-2\\
c([2i+1,n-1],[2i+2,n-1]) &= 2 \quad \forall 1 \leq i \leq \tfrac{n}{2}-2\\
c([n-2,n-1],[n-1,n-1]) &= 3
\end{aligned}$$

Free-Riders in Steiner Tree Cost-Sharing Games*

Paolo Penna and Carmine Ventre

Dipartimento di Informatica ed Applicazioni "R.M. Capocelli",
Università di Salerno, via S. Allende 2,
I-84081 Baronissi (SA), Italy
{penna, ventre}@dia.unisa.it

Abstract. We consider cost-sharing mechanisms for the Steiner tree game. In this well-studied *cooperative* game, each *selfish* user expresses his/her willingness to pay for being connected to a source node s in an underlying graph. A *mechanism* decides which users will be connected and divides the cost of the corresponding (optimal) Steiner tree among these users (*budget balance* condition). Since users can form *coalitions* and misreport their willingness to pay, the mechanism must be *group strategyproof*: even coalitions of users cannot benefit from lying to the mechanism.

We present new *polynomial-time* mechanisms which satisfy a standard set of axioms considered in the literature (i.e., budget balance, group strategyproofness, voluntary participation, consumer sovereignty, no positive transfer, cost optimality) and consider the *free riders* issue recently raised by Immorlica *et al.* [SODA 2005]: it would be desirable to avoid users that are connected for free. We also provide a number of negative results on the existence of *polynomial-time* mechanisms with certain guarantee on the number of free riders. Finally, we extend our technique and results to a variant considered by Biló *et al.* [SPAA 2004] with applications to *wireless* multicast cost sharing.

1 Introduction

Consider the typical scenario in which a set U of n users wishes to *jointly* buy a certain service from some service providing company. Each user $i \in U$ has a *private* value v_i representing his/her willingness to pay for the service offered: v_i is the maximum amount of money that user i is willing to pay for the service or how much he/she would benefit from getting the service. The service provider must then develop a so called *mechanism*, that is, a policy for deciding (i) which users should be serviced and (ii) the price that each of them should pay for getting the service. Developing a fair and economically viable cost-sharing mechanisms is a central problem in *cooperative* game theory with many practical applications (see e.g [18]). In particular, due to its application to multicast and

* Work supported by the European Project IST-2001-33135, Critical Resource Sharing for Cooperation in Complex Systems (CRESCCO).

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 231–245, 2005.

bandwidth allocation, *Steiner tree games* (and some variants) received a lot of attention [3, 7, 2, 15]. In this game(s), a network $G = (U \cup \{s\}, E, c)$ is given, where $U = \{1, 2, \ldots, n\}$ corresponds to the set of users and s is a source node. The weight c_e of an edge $e = (i, j) \in E$ denotes the cost of connecting i to j. The (minimum) cost $C(S)$ required to connect a subset S of users to the source is the cost of a (optimal) Steiner tree containing s and S.

An important class of mechanisms is the class of *budget-balanced* mechanisms, that is, the sum of the prices charged to all users is equal to the overall cost $C(S)$ of providing the service to the subset S of users that are selected for being serviced. Observe that, given a subset of users $S \subset U$, computing the optimal cost $C(S)$ is NP-hard [5]. Also, practical applications require the mechanism to output the optimal tree connecting S to s.

In addition, users cannot be assumed to be altruistic nor obedient. Therefore, each user is considered a *selfish agent* reporting some bid value b_i (possibly different from v_i); the true value v_i is *privately known* to agent i. Based on the *reported* values $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ a *mechanism* $M = (A, P)$ uses algorithm A to compute (i) a subset $S(\mathbf{b})$ of users and (ii) a Steiner tree $T(\mathbf{b})$ connecting s to $S(\mathbf{b})$ in G. Then, according to the payment vector $P = (P^1, P^2, \ldots, P^n)$, each user $i \in S(\mathbf{b})$ is charged an amount of money equal to $P^i(\mathbf{b})$. Selfish agents are assumed to be rational and thus, knowing the mechanism M, an user i could report $b_i \neq v_i$ whenever this increases his/her *utility*: given the bids $\mathbf{b}_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$ of the other agents, the utility function of user i is defined as

$$u_i^M(b_i, \mathbf{b}_{-i}) := \begin{cases} v_i - P^i(\mathbf{b}) & \text{if } i \in S(\mathbf{b}), \\ 0 & \text{otherwise.} \end{cases}$$

In [15] we provided the first mechanism for the Steiner tree game which meets all of the following axioms (previously considered in e.g. [13, 7]):

Cost Optimality (CO). Let $C_{\mathsf{opt}}(S(\mathbf{b}))$ denote the minimum cost required to service all users in $S(\mathbf{b})$. We require that the computed solution is an optimal Steiner tree for connecting the set $S(\mathbf{b})$ to the source s, that is, $\mathsf{COST}(T(\mathbf{b})) = C_{\mathsf{opt}}(S(\mathbf{b}))$.

No Positive Transfer (NPT). No user receives money from the mechanism, i.e., $P^i(\cdot) \geq 0$.

Voluntary Participation (VP). We never charge an user an amount of money greater than her *reported* valuation, that is, $\forall b_i, \forall \mathbf{b}_{-i}\ b_i \geq P^i(b_i, \mathbf{b}_{-i})$. In particular, a user has always the option to not pay for being connected if he/she is not interested. Moreover, $P^i(\mathbf{b}) = 0$, for all $i \notin S(\mathbf{b})$, i.e., only the users getting connected will pay.

Consumer Sovereignty (CS). Every user is guaranteed to get the service if she reports a high enough valuation, that is, $\forall \mathbf{b}_{-i}$, $\exists \overline{b}_i$ such that $i \in S(\overline{b}_i, \mathbf{b}_{-i}) = 1$.

Budget Balance (BB).

1. **Cost recovery.** The cost of the computed solution is recovered from all the users being serviced, that is,

$$\sum_{i \in S(\mathbf{b})} P^i(\mathbf{b}) \geq \mathsf{COST}(T(\mathbf{b})).$$

2. **Competitiveness.** No surplus is created, that is,

$$\sum_{i \in (\mathbf{b})} P^i(\mathbf{b}) \not> \mathsf{COST}(T(\mathbf{b})).$$

If some surplus were created, then a competitor may offer the same service at a better price.

Group Strategyproofness (GSP). We require that a user $i \in U$ that misreport her valuation (i.e., $b_i \neq v_i$) cannot improve her utility nor improve the utility of other users without worsening her own utility (otherwise, a coalition C containing i would secede). Consider a coalition $C \subseteq U$ of users. Let $b_j = v_j$ for all $j \notin C$. Let $\mathbf{b}_C$ and $\mathbf{b}_{-C}$ denote the bid vectors of those users in C and in $U \setminus C$, respectively. The group strategyproofness requires that, if the inequality

$$u_i^M(\mathbf{b}_C, \mathbf{v}_{-C}) \geq u_i^M(\mathbf{v}_C, \mathbf{v}_{-C}) \tag{1}$$

holds for all $i \in C$, then it must hold with equality for all $i \in C$ as well.

It is easy to see that, since the above property must be fulfilled for every possible $\mathbf{v}_{-C}$, then the GSP condition can be restated by replacing $\mathbf{v}_{-C}$ by $\mathbf{b}_{-C}$. In particular, the special case of $C = \{i\}$ yields the weaker notion of *strategyproofness* (or *truthfulness with dominant strategies*): for every user i, $\forall b_i$ and $\forall \mathbf{b}_{-i}$, it holds that $u_i^M(v_i, \mathbf{b}_{-i}) \geq u_i^M(b_i, \mathbf{b}_{-i})$.

It is worth observing that the mechanism in [15] runs in *polynomial time*, in spite of the NP-hardness of the problem of computing an optimal tree for a *given* set of terminals [5]. Intuitively, the mechanism in [15] is always able to pick a set $S(\mathbf{b})$ for which the optimal Steiner tree can be computed in polynomial time, thus ensuring the CO property.

Recently, Immorlica *et al* [6] considered cost-sharing games under the additional constraint of no *free riders*: no user should be serviced for free. This work, among other results, contains a general scheme for obtaining mechanisms with no free riders and satisfying all axioms above. For the Steiner tree game, their approach *cannot* lead to polynomial time mechanism, unless P = NP. By contrast, our polynomial-time mechanism in [15] is far from the no-free-riders condition: a single user will pay for the cost of the whole tree servicing the selected users S and thus all other users are free riders.

In this work we investigate the existence of mechanisms for the Steiner tree game (and some of its variants) that possibly reduce the number of free riders, still running is polynomial time.

1.1 Previous Work and Our Contribution

The breakthrough paper by Moulin and Shenker [13, 12] provided a natural and powerful technique for building group-strategyproof mechanisms based on the following tool: a *cost-sharing* function $\xi(\cdot)$ which specifies, for each subset S of users, how the cost $C(S)$ is shared among them, that is, $\forall S \subseteq U$, $\sum_{i \in S} \xi(S, i) = C(S)$. They indeed considered a natural scheme for building a mechanism $M(\xi)$ depending on the cost-sharing function $\xi(\cdot)$ (see Fig. 1), and proved that mechanism $M(\xi)$ is group strategyproof if the function $\xi(\cdot)$ is *cross-monotonic*, that is, for all $S' \subset S \subseteq U$, and for all $i \in S'$, if holds that $\xi(S', i) \geq \xi(S, i)$.

Mechanism $M(\xi)$

1. S is initialized to U;
2. If there exists an user i in S with $v_i < \xi(S, i)$ then drop i from S. Keep repeating this step, in arbitrary order, until for every user i in S, $v_i \geq \xi(S, i)$;
3. Charge each user i an amount equal to $P^i(\mathbf{b}) := \xi(S, i)$.

Fig. 1. A general scheme to build a mechanism starting from a cost-sharing function $\xi(\cdot)$ [13]

The existence of such functions was known for the MST game in a work by Kent and Skorin-Kapov [8]. Jain and Vazirani [7] provided a more general technique for building cross-monotonic cost-sharing methods for the MST game and proved that this technique yields a polynomial-time 2-approximate budget-balanced (BB) mechanism for the Steiner tree game. [1] The technique by Jain and Vazirani [7] is based on a non-trivial use of primal-dual algorithms and inspired several works which obtain polynomial-time α-approximate BB mechanisms for other cost-sharing games (namely, metric TSP [7], facility location [14], single-source rent-or-buy [14, 10], wireless multicast [2], Steiner forest [9]).

In a recent work, Immorlica *et al* [6] provided a number of lower bounds on the approximation factor α that cross-monotonic functions can achieve for some cost-sharing games. These lower bounds do not apply in general since mechanisms *not* using cross-monotonic cost-sharing functions may exist [15, 6]. In particular, in [15] we provide a polynomial-time mechanism for the Steiner tree game which achieves all axioms above. Unfortunately, this mechanism is far from the no-free-rider condition since it always charges the cost to a single user. [2]

[1] An α-approximate BB mechanism guarantees (only) that $\mathsf{COST}(T(\mathbf{b})) \leq \sum_i P^i(\mathbf{b}) \leq \alpha \cdot C_{\mathsf{opt}}(S(\mathbf{b}))$.

[2] A similar mechanism for general cost-sharing games has been also presented in [6], though their work does not investigate computational issues for the Steiner tree game.

In this work we provide a new mechanism which guarantees that, given its computed multicast tree T and the subset S_T of users connected to s, there are at most $|S_T| - |\mathsf{leaves}(T)|$ free riders, with $\mathsf{leaves}(T)$ being the set of leaves of tree T. This mechanism still runs in polynomial time and satisfies all axioms above. We also achieve similar results for the wireless multicast game considered in [2, 15]: for this game we obtain α-approximate BB mechanism with the same bound on the number of free riders. The factor α is the same of a mechanism in [15] which, however, has $|S_T| - 1$ free riders. To the best of our knowledge, this factor α is the best known for this game.

Since, in the worst case, our mechanisms yield $|S_T| - 1$ free riders, we investigate the existence of polynomial-time mechanisms having a better guarantee. We first prove that the scheme in [6] for budget-balanced mechanism does not apply to either of our problems and, in the worst case, has the same bad guarantee (i.e., $|S_T| - 1$ free riders). This negative result applies also to exponential-time mechanisms and it is due to the fact that the optimal cost function of our games are not subadditive[3], as required in [6]. Further, we show that *any* budget-balanced mechanism which has no free riders must solve an NP-hard problem. The same negative result holds for α-approximate BB mechanisms for the wireless multicast game, for some $\alpha > 0$. In particular, the $(1+\epsilon)$-approximate BB mechanism given in [6] turns out to be intractable for this game, for small $\epsilon > 0$.

It is worth observing that, under certain hypothesis, α-approximate BB mechanisms with no free riders do not exist for the Steiner tree game, for $\alpha < 2$. This follows from previous results by Immorlica *et al* [6] and van Zwam [17]. The mechanism proposed here satisfies these additional hypothesis and, therefore, free riders cannot be avoided.

1.2 Preliminaries and Notation

Consider a graph $G = (U \cup \{s\}, E, c)$ where the set of terminals coincides with the set of users U. Given a terminal set $S \subseteq U$, we let $ST^*(S)$ denote the minimum-cost tree connecting s to the set S. The tree $MST(S)$ is (any of) the minimum spanning tree(s) over the subgraph of G induced by all and only the vertices in $S \cup \{s\}$. We consider any tree T connecting S to s as rooted at s and we denote by $\mathsf{leaves}(T)$ the set of leaves resulting from this orientation. We consider every such a tree as the set of its directed edges (a, b), where a is the parent of b. For any node a of tree T, we define $\mathsf{leaves}(T, a)$ as the set of all leaf-nodes that are descendent of a.

A *cost-sharing method* for a cost function $C(\cdot)$ is a function $\xi(\cdot)$ which distributes this cost to the users that get the service. The function $\xi(\cdot)$ takes two arguments: a set of users S and a user i and returns a *nonnegative* real number satisfying the following:

$$\text{if } i \notin S \text{ then } \xi(S, i) = 0 \text{ and} \tag{2}$$

[3] A cost function $C(\cdot)$ is subadditive if $C(S \cup \{i\}) > C(S)$, for every $S \subset U$, $i \in U \setminus S$.

$$\sum_{i \in S} \xi(S,i) = C(S). \tag{3}$$

A *β-cost-sharing method* $\xi(\cdot)$ satisfies Eq. 2 and the following relaxation of Eq. 3: $C_{\mathsf{A}}(S) \leq \sum_{i \in S} \xi(S,i) \leq \beta \cdot C_{\mathsf{A}}(S)$. Given a function $\xi : 2^U \times U \to \mathbf{R}^+ \cup \{0\}$, we define $\mathcal{P}_0^\xi := U$, and $\mathcal{P}_j^\xi := \{S \setminus \{i\} |\ S \in \mathcal{P}_{j-1}^\xi \wedge \xi(S,i) > 0\}$. Intuitively, $\mathcal{P}_j^\xi$ contains the family of all possible sets S that the scheme in Fig. 1 can generate after j users have been dropped. Thus, the set $\mathcal{P}^\xi := \bigcup_{j \geq 0} \mathcal{P}_j^\xi$ contains all possible output sets $S(\mathbf{b})$ of $M(\xi)$. A function $\xi : 2^U \times U \to \mathbf{R}^+ \cup \{0\}$ is *self cross-monotonic* if, for every $S, S' \in \mathcal{P}^\xi$ with $S' \subset S$, it holds that $\xi(S',i) \geq \xi(S,i)$, for every $i \in S'$.

A mechanism M is *upper continous* if, fixed a vector $\mathbf{b}_{-i}$, if user i is serviced for every $b_i > b$, then it is also serviced for $b_i = b$. Clearly, the mechanism $M(\xi)$ is upper continous. Let A denote an algorithm that, given a set $S \subseteq U$, returns a tree connecting S to s. We plug this algorithm into the scheme in Fig. 1 by adding a final step in which a tree $T(\mathbf{b}) := \mathsf{A}(S(\mathbf{b}))$ is output, thus obtaining a mechanism $M_{\mathsf{A}}(\xi)$ for the Steiner tree game. This defines a cost function $C_{\mathsf{A}}(\cdot)$ satisfying $C_{\mathsf{A}}(S(\mathbf{b})) = \mathsf{COST}(\mathsf{A}(S(\mathbf{b})))$, for every subset $S(\mathbf{b}) \in \mathcal{P}^\xi$.

The following result provides an useful tool for building polynomial-time (approximate) budget-balanced group strategyproof mechanisms:

Theorem 1 ([15]). *For any optimal (respectively, α-approximation) algorithm A and any self cross-monotonic β-cost-sharing method $\xi(\cdot)$ for $C_{\mathsf{A}}(\cdot)$, the mechanism $M_{\mathsf{A}}(\xi)$ is group strategyproof, β-approximate BB (respectively, $\alpha\beta$-approximate BB) and satisfies NPT, VP and CS. Moreover, $M_{\mathsf{A}}(\xi)$ runs in polynomial time if A and $\xi(\cdot)$ are polynomial time.*

Observe that, if A is optimal in $\mathcal{P}^\xi$ only (i.e., $C_{\mathsf{A}}(S) = C_{\mathsf{opt}}(S)$, for every $S \in \mathcal{P}^\xi$) then mechanisms $M_{\mathsf{A}}(\xi)$ and $M_{\mathsf{opt}}(\xi)$ will output exactly the same solutions, for any optimal algorithm opt. Hence, the following holds:

Theorem 2. *Let $\xi(\cdot)$ be a self-cross monotonic cost-sharing $\xi(\cdot)$ for $C_{\mathsf{A}}(\cdot)$ and let A be optimal in $\mathcal{P}^\xi$. Then, the mechanism $M_{\mathsf{A}}(\xi)$ is group strategyproof, budget-balanced and satisfies CO, NPT, VP and CS. Moreover, $M_{\mathsf{A}}(\xi)$ runs in polynomial time if A and $\xi(\cdot)$ are polynomial time. Finally, $M_{\mathsf{A}}(\xi)$ is upper continous.*

2 Steiner Tree Game

We will develop a cost-sharing method which charges all and only the leaf nodes of the computed solution T. In particular, each time one node is dropped, a new tree is generated by removing the corresponding leaf. The possible trees that this process can possibly generate starting from $MST(U)$ is defined as follows:

Definition 1. *We let $T_n = MST(U)$ and $\mathcal{T}_n = \{T_n\}$. Then, given $\mathcal{T}_{j+1}$, we define inductively $\mathcal{T}_j := \{T |\ T \cup \{(a,b)\} \in \mathcal{T}_{j+1} \wedge b \in \mathsf{leaves}(T \cup \{(a,b)\})\}$. Moreover, for any $T \in \mathcal{T}_j$, we let S_T denote the set of all nodes of tree T other than s.*

In the next section we prove that any of these trees is optimal, that is, its cost equals the cost of the optimal Steiner tree connecting s to the same set of nodes of the tree.

2.1 Cost Optimality

Observe that, by definition, a minimum spanning tree $MST(U)$ is also an optimal Steiner tree for the terminal set U, that is, $\mathsf{COST}(ST^*(U)) = \mathsf{COST}(MST(U))$. We next show that, starting from this tree, if we repeatedly remove any leaf node, then the tree T that we obtain remains an optimal Steiner tree for the set of nodes it contains:

Theorem 3. *For any $T \in \mathcal{T}_j$, $\mathsf{COST}(ST^*(S_T)) = \mathsf{COST}(MST(T))$, with $0 \leq j \leq n$.*

Proof. The proof is by induction on j, starting from $j = n$ down to $j = 0$.

Base step ($j = n$). By Def. 1 we obtain $T = T_n$ and $S_{T_n} = U$. In this case, since there are no Steiner points, then the theorem clearly holds.

Inductive step (from $j+1$ to j). Let $T \in \mathcal{T}_j$, thus implying that there exists an edge (a, b) such that $T^b := T \cup \{(a, b)\} \in \mathcal{T}_{j+1}$ and b is a leaf in T^b. Hence, $a \in S_T$ and $S_{T^b} = S_T \cup \{b\}$.

By contradiction assume $\mathsf{COST}(T) > \mathsf{COST}(ST^*(S_T))$. We will show that there exists a tree T' spanning S_{T^b} and whose cost is lower than the cost of $ST^*(S_{T^b})$, thus contradicting the optimality of $ST^*(S_{T^b})$. The new tree T' is constructed as follows:

$$T' := \begin{cases} ST^*(S_T) \cup \{(a, b)\} & \text{if } b \notin ST^*(S_T), \\ ST^*(S_T) & \text{otherwise.} \end{cases}$$

We thus obtain

$$\mathsf{COST}(T') \leq \mathsf{COST}(ST^*(S_T)) + c_{(a,b)} < \mathsf{COST}(T) + c_{(a,b)} = \mathsf{COST}(T^b).$$

Since $a \in S_T$, then T' is a tree spanning $S_T \cup \{b\} = S_{T^b}$. By inductive hypothesis, we have $\mathsf{COST}(T^b) = \mathsf{COST}(ST^*(S_{T^b}))$, and the above inequality yields $\mathsf{COST}(T') < \mathsf{COST}(ST^*(S_{T^b}))$. This contradicts the optimality of $ST^*(S_{T^b})$.

This completes the proof.

2.2 Cost-Sharing Function

We first define a function $\xi_T(\cdot)$ that shares the cost of T among its leaves:

Definition 2. *Given a tree T and any $a \in \mathsf{leaves}(T)$, we let*

$$\xi_T(a) := \sum_{(u,v) \in T : a \in \mathsf{leaves}(T,u)} \frac{c_{(u,v)}}{|\mathsf{leaves}(T, u)|}. \tag{4}$$

Lemma 1. *For any tree T, it holds that* $\sum_{a\in\mathsf{leaves}(T)} \xi_T(a) = \mathsf{COST}(T)$.

Proof. Since for every $a \in \mathsf{leaves}(T)$ there exists $(u,v) \in T$ such that $a \in \mathsf{leaves}(T,u)$, we simply observe that

$$\sum_{(u,v)\in T} \sum_{a\in\mathsf{leaves}(T,u)} \frac{c_{(u,v)}}{|\mathsf{leaves}(T,u)|} = \sum_{a\in\mathsf{leaves}(T)} \sum_{\substack{(u,v)\in T:\\ a\in\mathsf{leaves}(T,u)}} \frac{c_{(u,v)}}{|\mathsf{leaves}(T,u)|}$$

The left hand side is $\mathsf{COST}(T)$, while the right hand side is $\sum_{a\in\mathsf{leaves}(T)} \xi_T(a)$. The lemma thus follows.

We use the trees $\mathcal{T}_j$ and the associated functions $\xi_T(\cdot)$ for defining a self cross-monotonic cost-sharing function $\xi_{\mathsf{leaves}}(\cdot)$ as follows:

$$\xi_{\mathsf{leaves}}(S_T, a) = \begin{cases} \xi_T(a) \text{ if } a \in \mathsf{leaves}(T), \\ 0 \qquad \text{otherwise,} \end{cases} \tag{5}$$

for every $T \in \mathcal{T}_j$, with $0 \le j \le n$.

Lemma 2. *Let $\xi = \xi_{\mathsf{leaves}}$. Then $\mathcal{P}_j^{\xi} = \bigcup_{T\in\mathcal{T}_j} S_T$, for every $0 \le j \le n$.*

Theorem 4. *The function $\xi_{\mathsf{leaves}}(\cdot)$ is self cross-monotonic.*

Proof. It is easy to see that, since $\xi_{\mathsf{leaves}}(\cdot)$ is non-zero only for the leaf nodes, then starting from $T_n = MST(U)$, at each step the mechanism $M(\xi_{\mathsf{leaves}})$ will consider a tree $T \in \mathcal{T}_j$ and remove some leaf b. Let $T' = T \setminus (a,b)$ denote the new tree obtained in this way. We prove that $\xi_{\mathsf{leaves}}(S_T, i) \le \xi_{\mathsf{leaves}}(S_{T'}, i)$, for every $i \in S_{T'}$. Since $S_{T'} = S_T \setminus \{b\}$, it holds that $i \ne b$. For every $(u,v) \in T'$, $|\mathsf{leaves}(T,u)| \ge |\mathsf{leaves}(T',u)|$. Moreover, by definition of T', if $i \in \mathsf{leaves}(T,u)$ for some edge (u,v), then $i \in \mathsf{leaves}(T',u)$. Therefore, if $i \in \mathsf{leaves}(T)$, we obtain

$$\begin{aligned} \xi_{\mathsf{leaves}}(S_T, i) &= \sum_{\substack{(u,v)\in T:\\ i\in\mathsf{leaves}(T,u)}} \frac{c_{(u,v)}}{|\mathsf{leaves}(T,u)|} \le \sum_{\substack{(u,v)\in T:\\ i\in\mathsf{leaves}(T,u)}} \frac{c_{(u,v)}}{|\mathsf{leaves}(T',u)|} \\ &= \sum_{\substack{(u,v)\in T':\\ i\in\mathsf{leaves}(T',u)}} \frac{c_{(u,v)}}{|\mathsf{leaves}(T',u)|} = \xi_{\mathsf{leaves}}(S_{T'}, i). \end{aligned}$$

Otherwise, that is $i \notin \mathsf{leaves}(T)$, it simply holds $\xi_{\mathsf{leaves}}(S_T, i) = 0 \le \xi_{\mathsf{leaves}}(S_{T'}, i)$.

Now consider any two trees $T \in \mathcal{T}_j$ and $T' \in \mathcal{T}_{j-k}$ with $S_{T'} \subset S_T$. By definition, there exists a sequence of trees $T_1, T_2, \ldots, T_{k+1}$, with $T_1 = T$ and $T_k = T'$, which is obtained by repeatedly removing some leaf node of T_l, with $1 \le l \le k-1$. The above argument thus yields $\xi_{\mathsf{leaves}}(S_{T_1}, i) \le \xi_{\mathsf{leaves}}(S_{T_2}, i) \le \cdots \le \xi_{\mathsf{leaves}}(S_{T_k}, i)$. The theorem thus follows from Lemma 2.

2.3 Analysis

We first observe that Lemma 2 and Theorem 3 imply that MST is optimal on $\mathcal{P}^{\xi_{\mathsf{leaves}}}$. Hence, by applying Theorem 2 with $\xi = \xi_{\mathsf{leaves}}$, we obtain the following result:

Corollary 1. *The Steiner tree game admits a mechanism $M_{MST}(\xi)$ which is polynomial-time, group strategyproof, budget-balanced and satisfies CO, NPT, VP and CS. The mechanism is upper continous and garantees that, if T is the tree given in output, then there are at most $|S_T| - |\mathsf{leaves}(T)|$ free riders.*

3 Wireless Multicast Game

A variant of the Steiner tree problem considered in [2, 15] is the *wireless multicast* which is defined as follows. Each node of the graph G corresponds to a station of an ad-hoc network. Stations are located on a c-dimensional Euclidean plane and, given the distance $d(i, j)$ between i and j, the cost of connecting i to j is $c_{(i,j)} := d(i, j)^\gamma$, for some $\gamma > 1$. This quantity represents the power that station i must spend to transmit directly to j. Thus, given a multicast tree T, its cost is the *overall power consumption*, that is,

$$\mathsf{POW}(T) := \sum_{i \in U \cup \{s\}} \max_{(i,j) \in T} \{c_{(i,j)}\},$$

that is, every node contributes as the cost of its longest outgoing edge in T. The game is thus defined as the Steiner tree game with the only difference that the cost function $\mathsf{COST}(\cdot)$ is replaced by the function $\mathsf{POW}(\cdot)$ above. As in [2, 15], we will consider $\gamma \geq c$, since for $\gamma < c$ no approximation algorithm in known; moreover, in many applications $\gamma \geq 2$ and stations are located on the plane (i.e., $c = 2$).

Since $\mathsf{POW}(T) < \mathsf{COST}(T)$ whenever T as at least two leaves, if we apply the payment scheme for the Steiner tree game, the mechanism will create some *surplus*, that is, users will pay more than the cost. We next modify the payment scheme so to avoid this.

Given a tree T, and any node i, let $e_1(i), e_2(i), \ldots, e_k(i)$ denote the list of nodes in T outgoing from i and satisfying $c_{e_j(i)} \leq c_{e_{j+1}(i)}$ (ties are broken arbitrarily). We define a function $w(\cdot)$ which weights the edges of T according to their contribution to $\mathsf{POW}(T)$. In particular, let $\mathsf{mc}(e_j(i))$ denote the *marginal contribution*[4] of edge $e_j(i)$, that is,

$$\mathsf{mc}(e_j(i)) := \begin{cases} c_{e_j(i)} - \max\{c_{e_l(i)} |\ c_{e_l(i)} < c_{e_j(i)}\} & \text{if } j > 1, \\ c_{e_j(i)} & \text{otherwise.} \end{cases} \tag{6}$$

[4] A similar concept is employed in [2] for defining the Shapley value of a wireless multicast tree.

Finally, defined

$$\mathsf{equal}(e_j(i)) := |\{e_l(i) | c_{e_l(i)} = c_{e_j(i)}\}|$$

we let

$$w_{e_j(i)} := \mathsf{mc}(e_j(i))/\mathsf{equal}(e_j(i)). \tag{7}$$

By definition, $\sum_{j=1}^{k} w_{e_j(i)} = c_{e_k(i)}$, thus implying $\mathsf{POW}(T) = \sum_{e \in T} w_e$. We can share this cost by considering the graph G with edge costs c_e replaced by w_e, for every $e \in T$. This idea leads to the following result:

Theorem 5. *Let w_e be defined as in Eq. 7 with respect to $T = MST(U)$. Also let $\xi_{\mathsf{wireless}}(\cdot)$ be defined as in Eq.s 4-5 by replacing c_e with w_e, for every $e \in T$. Then, the function $\xi_{\mathsf{wireless}}(\cdot)$ is a self-cross monotonic cost-sharing method for $C_{\mathsf{A}}(S) := \mathsf{POW}(MST(S))$.*

We can modify the mechanism for the Steiner tree game by replacing $\xi_{\mathsf{leaves}}(\cdot)$ by $\xi_{\mathsf{wireless}}(\cdot)$, thus obtaining a polynomial-time mechanism $M_{MST}(\xi_{\mathsf{wireless}})$ which outputs a tree $T \in \mathcal{T}$ (see Def. 1) and that recovers the corresponding cost $\mathsf{POW}(T)$. In [4] the authors proved that, for any $\gamma \geq \delta$, the total weight of a MST over a set S of points is at most $(3^c - 1)$ times the cost of the optimal wireless broadcast tree for S. (For $c = 2 \leq \gamma$ the constant has been improved down to 7.5 [4].) Thus, using an argument similar to [2, 15], Theorem 3 yields the following result:

Corollary 2. *The wireless multicast game admits a polynomial-time mechanism $M_{MST}(\xi)$ which is group strategyproof, $(3^c - 1)$-approximate BB and satisfies NPT, VP and CS. The mechanism is upper continous and guarantees that, if T is the tree given in output, then there are at most $|S_T| - |\mathsf{leaves}(T)|$ free riders. Additionally, for $c = 2$, mechanism $M_{MST}(\xi_{\mathsf{wireless}})$ is 7.5-approximate BB.*

The above result improves over the mechanism in [15] since in this mechanism there are always $|S_T| - 1$ free riders. In the next section we compare our mechanism with other mechanisms.

4 Negative Results and Open Questions

Immorlica *et al* [6] proposed two mechanisms for avoiding free riders in a cost-sharing game with cost function $C(\cdot)$. The first mechanism is budget-balanced and works for *subadditive* functions $C(\cdot)$:

Mechanism IMM-BUDGET-BALANCE

1. Initialize the set S of serviced users to the empty set and the amount of money m that is already charged to 0;
2. For i from 0 to n, do the following: if $b_i \geq \min\{C(\{i\}), C(S \cup \{i, \ldots, n\}) - m\}$, then include i in S, and charge him/her $\min\{C(\{i\}), C(S \cup \{i, \ldots, n\}) - m\}$ (therefore, m will be increased by this quantity).

The following fact shows that, in our games, the optimal cost function is *not* subadditive and, more importantly, mechanism IMM-BUDGET-BALANCE does *not* guarantee any bound on the number of free riders:

Fact 6. *There exists an instance of the Steiner tree game for which the mechanism* IMM-BUDGET-BALANCE *yields* $n-1$ *free riders. The same result holds for the wireless multicast game.*

Proof. Consider a clique K_n of n nodes and let any of its edges have cost 0. We build a graph G by connecting a new node s to every node of K_n with an edge of cost 1. Then, for every $S \subseteq U$, with $S \neq \emptyset$, $C(S) = 1$. Hence, mechanism IMM-BUDGET-BALANCE charges 0 to all but one user in the final set S.

A similar argument applies to the following instance of the wireless multicast game: consider the star G connecting s to $n \geq 2$ nodes at distance 1 from s. In this case, there will be, in the worst case, $n-1$ free riders.

We also mention that, for the Steiner tree game, a known result by Megiddo [11], combined with a characterization of upper continous budget-balanced mechanisms without free riders [6–Th. 4.3], implies that the above (upper continous) mechanism cannot guarantee the no-free-rider condition even if edges have non-zero weight.

We next show that, unless $\mathsf{P} = \mathsf{NP}$, there exist no polynomial-time mechanism satisfying all axioms above plus the no-free-rider condition.

Theorem 7. *Let* $C_{\mathsf{opt}}(\cdot)$ *be a cost function which is* NP*-hard (to approximate within a factor* α*). Then, no polynomial-time strategyproof mechanism* M *guarantees no free riders and satisfies NPT, VP, CS and (*α*-approximate) BB, unless* $\mathsf{P} = \mathsf{NP}$.

Proof Sketch. The strategyproofness, NPT, BB (no surplus part) and CS conditions imply that, for every $S \subseteq U$, there exists a bid vector $\mathbf{b}^S = (\mathbf{b}_S, \mathbf{0}_{-S})$ such that $S(\mathbf{b}^S) \supseteq S$. The VP and the no-free-rider condition imply that $S(\mathbf{b}^S) = S$. Hence, the (α-approximate) BB condition implies that M is able to compute (an α-approximation of) the optimum $C_{\mathsf{opt}}(S)$ in polynomial time, thus implying $\mathsf{P} = \mathsf{NP}$. (See Appendix 5 for the full proof of this theorem.) □

Since the Steiner tree problem cannot be approximated within some factor $\alpha > 1$, then it is not possible to have polynomial time α-approximate BB mechanisms and no free riders. Hence, the following mechanism for $(1+\epsilon)$-approximate BB mechanisms [6] cannot run in polynomial time for small $\epsilon > 0$:

Mechanism IMM-$(1+\epsilon)$-APPROXIMATE-BB

1. Drop all users i such that $b_i \leq \delta$ and let $R = \{r_1, r_2, \ldots, r_{|S|}\}$ be the (arbitrarily) ordered set of remaining users; /* the value of δ depends on $\epsilon > 0$ */
2. Find the first user $r_i \in R$ such that $b_{r_i} \geq C(\{r_i, r_{i+1}, \ldots, r_{|S|}\}) - n\delta$; The set $S := \{r_i, r_{i+1}, \ldots, r_{|S|}\}$ is serviced, user i pays $C(S) - n\delta$, and every body else in S pays δ.

Observe that, for the wireless multicast game, the mechanism above would be polynomial-time only if a polynomial-time $(1+\epsilon)$-approximation algorithm for the wireless *multicast* optimization problem exists. To the best of our knowledge, the best factor is achieved by a combination of the best known Steiner tree algorithm [16] with the results in [4]. This combination (see [1]) yields an upper bound which is *larger* than the approximation factor $(3^c - 1)$ for the wireless broadcast [4]. Whether such bounds are tight is an interesting open problem. Below we mention other questions that remain open.

4.1 Open Questions

In view of these negative results, one may try to improve our mechanisms along two directions: (i) decrease the number of free riders and/or (ii) improve the approximation factor of the mechanism for wireless multicast game.

As already mentioned, a result by Immorlica *et al* [6] characterizes the class of upper continous α-approximate budget-balanced mechanisms with no free riders as the mechanisms which can be obtained using α-cross-monotonic cost-sharing methods. Hence, the lower bounds on cross-monotonic cost-sharing methods [11, 2] imply that, for the two games considered here, mechanisms without free riders, if any, cannot be obtained directly from the scheme in Fig. 1 or from mechanism IMM-BUDGET-BALANCE (both of them being upper continous). For the Steiner tree game, a tight result by van Zwam [17], implies that the polynomial-time 2-approximate BB mechanisms by [7] are the best possible in the class of upper continous one.

Concerning polynomial-time mechanisms and the free rider issue, the following question is interesting to us: Is there any such mechanism which guarantees that a constant fraction of the serviced set S_T are not free riders? First of all, it is well-known that every node in G of degree 2 cannot be a Steiner point if G is metric. Hence, these nodes could also be charged and participate to the payments. Unfortunately, nodes that in tree $T_n = MST(U)$ have degree 2 *can* be Steiner points and their removal could lead to suboptimal solutions.

Generally speaking, the main question left open is to find a trade off between the number of free-riders and the approximation factor of the budget balance for polynomial-time mechanisms.

References

1. E. Althaus, G. Calinescu, I. Mandoiu, S. Prasad, N. Tchervenski, and A. Zelikovsky. Power efficient range assignment in ad-hoc wireless networks," submitted journal version. *Preliminary results in WCNC '03 and TCS '03*, 2003. Submitted journal version available at http://www.cs.iit.edu/∼calinesc/.
2. V. Biló, C. Di Francescomarino, M. Flammini, and G. Melideo. Sharing the cost of multicast transmissions in wireless networks. In *Proceedings of SPAA*, pages 180–187, 2004.
3. J. Feigenbaum, C.H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001.

4. M. Flammini, A. Navarra, R. Klasing, and S. Perennes. Improved approximation results for the minimum energy broadcasting problem. In *Proc. of the joint workshop on Foundations of mobile computing (DIALM-POMC)*, pages 85–91. ACM Press, 2004.
5. M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, 1979.
6. N. Immorlica, M. Mahdian, and V. Mirrokni. Limitations of cross-monotonic cost-sharing schemes. In *Proc. of the 16th Annual ACM Symposium on Discrete Algorithms (SODA)*, January 2005.
7. K. Jain and V.V. Vazirani. Applications of approximation algorithms to cooperative games. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–372, 2001.
8. K. Kent and D. Skorin-Kapov. Population monotonic cost allocation on MST's. In *Operational Research Proceedings KOI*, volume 43-48, 1996.
9. J. Könemann, S. Leonardi, and G. Schäfer. A group-strategyproof mechanism for Steiner forests. In *Proc. of the 16th Annual ACM Symposium on Discrete Algorithms (SODA)*. ACM Press, January 2005. To appear.
10. S. Leonardi and G. Schäfer. Cross-monotonic cost-sharing methods for connected facility location games. Technical Report MPI-I-2003-1-017, Max-Plankt-Institut für Informatik, September 2003. Also in Proc. of 5th ACM Conference on Electronic Commerce (EC), pages 242-243, ACM Press, 2004.
11. N. Megiddo. Cost allocation for Steiner trees. *Networks*, 8:1–6, 1978.
12. H. Moulin. Incremental cost sharing: characterization by coalition strategyproofness. *Social Choice and Welfare*, 16:279–320, 1999.
13. H. Moulin and S. Shenker. Strategyproof sharing of submodular costs: budget balance versus efficiency. 1997. http://www.aciri.org/shenker/cost.ps.
14. M. Pàl and É. Tardos. Strategy proof mechanisms via primal-dual algorithms. In *Proc. of the 44th FOCS*, 2003.
15. P. Penna and C. Ventre. More Powerful and Simpler Cost-Sharing Methods. In *Proceedings of the 2nd Workshop on Approximation and Online Algorithms (WAOA)*, number 3351 in LNCS, September 2004. To appear in the LNCS series.
16. A. Robins and Zelikovsky. Improved approximation Steiner tree in graphs. In *Proc. of the 11th Annual ACM Symposium on Discrete Algorithms (SODA)*, pages 770–779, 2000.
17. S. van Zwam. A lower bound on the cost recovery of the Steiner tree game with cross-monotonic cost shares. Technical Report 18-04, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2004.
18. H.P. Young. *Cost Allocation*, volume 2, chapter 34 of Handbook of Game Theory, volume 2, edited by R.J. Aumann and S. Hart, pages 1193–1235. Elsevier Science, 1994.

5 Proof of Theorem 7

We next show two useful lemmata which state some basic properties of strategyproof mechanisms. Basically, these results concern how changing one bid in the vector $\mathbf{b}$ can affect the outcome of the mechanism (i.e., the serviced set and the computed payments). Given a mechanism $M = (A, P)$, we let $\sigma_i^A(\mathbf{b}) = 1$

if and only if $i \in S(\mathbf{b})$, with $S(\mathbf{b})$ being the subset of users that algorithm A decides to service on input $\mathbf{b}$.

The following lemma states "steady" conditions on strategyproof mechanisms:

Lemma 3 (keep). *Let $M = (A, P)$ be a strategyproof mechanism and let $S(\mathbf{b})$ denote the set of users serviced on input $\mathbf{b}$. Then, the following conditions must hold:*

$$\sigma_i^A(b_i, \mathbf{b}_{-i}) \Rightarrow \forall b_i' > b_i,\ i \in S(b_i', \mathbf{b}_{-i}); \tag{8}$$

$$\sigma_i^A(b_i, \mathbf{b}_{-i}) = \sigma_i^A(b_i', \mathbf{b}_{-i}) \Rightarrow\ P^i(b_i, \mathbf{b}_{-i}) = P^i(b_i', \mathbf{b}_{-i}). \tag{9}$$

Proof. (Eq. 8). By contradiction, if $\sigma_i^A(b_i', \mathbf{b}_{-i}) = 0$ and $\sigma_i^A(b_i, \mathbf{b}_{-i}) = 1$, for some $b_i' > b_i$ and for some $\mathbf{b}_{-i}$, then consider the situation in which $v_i = b_i'$ and $\mathbf{v}_{-i} = \mathbf{b}_{-i}$. In this case, we obtain

$$v_i \cdot \sigma_i^A(b_i, \mathbf{v}_{-i}) - P^i(b_i, \mathbf{v}_{-i}) \geq v_i - b_i = b_i' - b_i > 0 \geq v_i \cdot \sigma_i^A(v_i, \mathbf{v}_{-i}) - P^i(v_i, \mathbf{v}_{-i}),$$

where the last inequality follows from the fact that $\sigma_i^A(v_i, \mathbf{v}_{-i}) = \sigma_i^A(b_i', \mathbf{v}_{-i}) = 0$ and from the NPT condition. This contradicts the fact that M is strategyproof.

(Eq. 9). By contradiction, let us assume that $\sigma_i^A(b_i, \mathbf{b}_{-i}) = \sigma_i^A(b_i', \mathbf{b}_{-i})$ and $P^i(b_i, \mathbf{b}_{-i}) < P^i(b_i', \mathbf{b}_{-i})$, for some b_i, b_i' and $\mathbf{b}_{-i}$. Consider the situation in which $v_i = b_i'$, thus implying

$$v_i \cdot \sigma_i^A(b_i, \mathbf{v}_{-i}) - P^i(b_i, \mathbf{v}_{-i}) > v_i \cdot \sigma_i^A(v_i, \mathbf{v}_{-i}) - P^i(v_i, \mathbf{v}_{-i}),$$

thus contradicting the fact that M is strategyproof.

The following lemma states that, if users "compete" with each other for being serviced, then the prices cannot be bounded from above by any constant:

Lemma 4 (drop). *Let $M = (A, P)$ a strategyproof mechanism satisfying NPT and CS. For every $\mathbf{b} = (b_1, \ldots, b_n)$, if there exists $i, j \in U$ and $\overline{b}_j$, such that*

$$\sigma_i^A(\mathbf{b}) = 1; \tag{10}$$

$$\sigma_i^A(\overline{\mathbf{b}}) = 0,\ \ \textit{with}\ \overline{\mathbf{b}} = (\overline{b}_j, \mathbf{b}_{-j}). \tag{11}$$

Then there exists a $\overline{b}_i$ such that $P^i(\mathbf{b}') \geq b_i$, where $\mathbf{b}' = (\overline{b}_i, \overline{\mathbf{b}}_{-i})$.

Proof. From the CS condition, there exists $\overline{b}_i$ such that $\sigma_i^A(\overline{b}_i, \overline{\mathbf{b}}_{-i}) = 1$, where $\overline{\mathbf{b}} = (\overline{b}_j, \mathbf{b}_{-j})$. By contradiction, assume that $P^i(\overline{b}_i, \overline{\mathbf{b}}_{-i}) < b_i$. Consider the case in which $v_i = b_i$ and $\mathbf{v}_{-i} = \overline{\mathbf{b}}_{-i}$. Because of the NPT and Eq. 11, it holds that $u_i^M(v_i, \mathbf{v}_{-i}) = u_i^M(b_i, \overline{\mathbf{b}}_{-i}) \leq 0$. Moreover, $u_i^M(\overline{b}_i, \mathbf{v}_{-i}) = u_i^M(\overline{b}_i, \overline{\mathbf{b}}_{-i}) = v_i - P^i(\overline{b}_i, \overline{\mathbf{b}}_{-i}) > 0$, thus contradicting the fact that $M = (A, P)$ is strategyproof.

The above lemma easily implies the following result.

Lemma 5. *Let $M = (A, P)$ a strategyproof mechanism satisfying NPT and CS. For every $S \subseteq U$ and for every $\overline{b} > 0$, there exists a vector $\mathbf{b}^S = (\mathbf{b}_S, \mathbf{0}_{-S})$ such that, if mechanism M returns a set $S(\mathbf{b}^S)$ of users with $S \not\subseteq S(\mathbf{b}^S)$, then there exists $i \in U$ such that $P^i(\mathbf{b}) > \overline{b}$.*

Proof. Simply increase the bids of users in S one by one, from 0 to a value $\overline{b}_i \geq \overline{b}$ such that the current user must be serviced. (This value exists because of the CS condition.) If at some point, a user $i \in S$ previously considered is dropped, because of Lemma 4, there exists a bid vector $\mathbf{b}'$ for which $P^i(\mathbf{b}') > \overline{b}$.

Let $M = (A, P)$ be a polynomial-time mechanism satisfying the hypothesis of Theorem 7. By contradiction, let M run in polynomial time. The above lemma and the (α-approximate) BB imply $S \subseteq S(\mathbf{b})$. The no-free-rider and VP conditions thus yield $S(\mathbf{b}) = S$. Hence, the (α-approximate) BB condition implies that M is able to compute (an α-approximation of) the optimum $C_{\mathsf{opt}}(S)$ in polynomial time, thus implying $\mathsf{P} = \mathsf{NP}$.

On the Feasibility of Gathering by Autonomous Mobile Robots

Giuseppe Prencipe

Dipartimento di Informatica,
L.go Bruno Pontecorvo, 3 – 56100, Pisa, Italy
prencipe@di.unipi.it

Abstract. Given a set of n autonomous mobile robots that can freely move on a two dimensional plane, they are required to gather in a position of the plane not fixed in advance (GATHERING PROBLEM). The main research question we address in this paper is: under which conditions this task can be accomplished by the robots? The studied robots are quite simple: they are anonymous, totally asynchronous, they do not have any memory of past computations, they cannot explicitly communicate among each other. We show that this simple task cannot be in general accomplished by the considered system of robots.

Keywords: Mobile Robots, Multiplicity Detection, Distributed Coordination, Distributed Models, Computability.

1 Introduction

In this paper, we consider a distributed system populated by a set of n autonomous and anonymous mobile robots that can freely and independently move on a plane: in particular, they do not obey to any central coordinator. The behavior of these robots is quite simple: each of them execute a cycle of sensing, computing, moving and being inactive. In particular, each robot is capable of sensing the positions of other robots in its surrounding, performing local computations on the sensed data, and moving towards the computed destination. The local computation is done according to a deterministic algorithm that takes in input the sensed data (i.e., the robots' positions), and returns a destination point towards which the executing robot moves. All robots execute the same algorithm. The main research focus is to understand which are the conditions that allow these robots to complete given tasks, such as exploring the plane or forming a pattern like a circle, and design, in case the task is solvable, to design the algorithm they have to execute.

In this paper we focus on the GATHERING problem: the robots are asked to meet in *finite time* in a point $\mathfrak{p}$ of the plane not determined in advance. In spite of its apparent simplicity, this problem has recently been tackled in several studies: in fact, several factors render this problem difficult to solve [3, 4, 5, 8, 10]. In particular, in all these studies, the problem has been solved only

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 246–261, 2005.

making some "extra" assumption on the capability of the robots. In particular, in [4, 5, 10] the robots must be able to detect whether a given point on the plane is occupied by one or more robots. In contrast, such an assumption is not used in [3], but it is assumed an unlimited amount of memory the robots can use (the robots are said to be *non-oblivious*). In [8], the robots are assumed to have only limited visibility (i.e., they can sense only a portion of the plane) and to share a compass. Recently [1], GATHERING has also been studied in the presence of faulty robots; another study has been devoted to design convergence solutions to the problem [6].

In this paper we aim to prove that GATHERING is in general impossible, if the nature of the robots is not changed, and no "extra" assumption is made on the capabilities of the robots. The results shown here, are based on the *basic* model usually adopted in the majority of the studies present in literature. In particular, we will use the features of CORDA, first presented in [7], and of the model from Suzuki *et al.* (here referred to as ATOM): the description of these models will be the focus of the next section. In Section 3 the impossibility of the GATHERING is presented.

2 Definitions

2.1 Autonomous Mobile Robots

In this section, we describe the CORDA model, that will be used to prove the impossibility of GATHERING. The robots we consider are modeled as devices with computational capabilities[1], that are equipped with motorial capabilities – allowing them to move on the plane – and sensorial capabilities that let them to observe the positions of the other robots in the plane, and form their *local view* of the world. The set of absolute positions[2] on the plane occupied by the robots at a given time instant is called a *configuration of the robots*.

The local view of each robot includes a unit of length, an origin, and a Cartesian coordinate system defined by the *directions* of two coordinate axes, identified as the x and y axis, together with their *orientations*, identified as the positive and negative sides of the axes.

The robots are able to sense the complete plane: we say they have *Unlimited Visibility*. The robots, however, can not distinguish whether there is more than one fellow on a given positions of the plane: we say that they cannot detect *multiplicity*. The case when the robots can sense just a portion of it (*Limited Visibility*) has been studied too [8]; in particular, each robot can sense up to at most a distance V.

[1] To our knowledge, nothing is ever mentioned on the computational power of the modeled robots. For the purpose of this paper, they can be considered as Turing-equivalent machines.

[2] i.e., with respect to an inertial reference frame.

During its life, each robot cyclically executes four *states*:

i. Wait. The robot is idle. A robot cannot stay indefinitely idle (see Assumption A2 below). At the beginning all the robots are in *Wait*.

ii. Look. The robot observes the world by activating its sensors which will return a *snapshot* of the positions of all other robots within the visibility range with respect to its local coordinate system. Each robot is viewed as a point, hence its position in the plane is given by its coordinates, and the result of the snapshot (hence, of the observation) is just a set of coordinates in its local coordinate system: this set forms the *view of the world* of r. More formally, the view of the world of r at time t is defined as the last snapshot taken at a time smaller than or equal to t.

iii. Compute. The robot performs a *local computation* according to a deterministic algorithm $\mathcal{A}$ (we also say that the robot *executes* $\mathcal{A}$). The algorithm is the same for all robots, and the result of the *Compute* state is a *destination point*. Since the robots are oblivious, then $\mathcal{A}$ can access only the set of robots' positions retrieved during the last *Look*.

iv. Move. If the point computed in the previous state is the current location of r, we say that r performs a *null movement*, and it does not move; otherwise it moves towards the point computed in the previous state. The robot moves towards the computed destination of an unpredictable amount of space, which is assumed neither infinite, nor infinitesimally small (see Assumption A1 below). Hence, the robot can only go towards its goal, but it cannot know how far it will go in the current cycle, because it can stop anytime during its movement[3]. The amount of space traveled by a robot during this state is also called the *length* of the move.

The sequence: *Wait - Look - Compute - Move* will be called a *computation cycle* (or briefly *cycle*) of a robot.

The (global) time that passes between two successive states of the same robot is finite but unpredictable. In addition, no time assumption within a state is made. This implies that the time that passes after the robot starts observing the positions of all others and before it starts moving is arbitrary, but finite. That is, the actual move of a robot may be based on a situation that was observed arbitrarily far in the past, and therefore it may be totally different from the current situation.

This assumption of *asynchronicity within a cycle* is important in a totally asynchronous environment, since each robot has enough time to perform its local computation; furthermore, in this way it is possible to model also different motorial speeds of the robots.

In the model, there are only two limiting assumptions about space and time. The first one refers to space.

[3] That is, a robot can stop before reaching its destination point, e.g. because of limits to the robot's motorial capabilities.

Assumption A1(Distance). *The distance traveled by a robot r in a move is not infinite. Furthermore, there exists an arbitrarily small constant $\delta_r > 0$, such that if the destination point is closer than δ_r, r will reach it; otherwise, r will move towards it of at least δ_r.*

As no other assumptions on space exist, the distance traveled by a robot in a cycle is unpredictable.

Similarly, to prove that the algorithms designed in CORDA terminate in finite time, the following assumption on the length of a computational cycle is made.

Assumption A2(Computational Cycle). *The amount of time required by a robot r to complete a computational cycle is not infinite. Furthermore, there exists a constant $\varepsilon_r > 0$ such that the cycle will require at least ε_r time.*

As no other assumption on time exists, the resulting system is *fully asynchronous* and the duration of each activity (or inactivity) is unpredictable. As a result, the robots do not have a common notion of time, robots can be seen while moving, and computations can be made based on obsolete observations.

The robots do not necessarily share the same $x-y$ coordinate system, and do not necessarily agree on the location of the origin (that we can assume, without loss of generality, to be placed in the current position of the robot), or on the unit distance. In general, there is no agreement among the robots on the chirality of the local coordinate systems (i.e., in general they do not share the same concept of where North, East, South, and West are).

The robots are totally *oblivious*; that is, the robots can only store the robots' positions retrieved in the last observation.

The robots are completely *autonomous*: no central control is needed. Furthermore they are *anonymous*, meaning that they are a priori indistinguishable by their appearance, and they do not (need to) have any kind of identifiers that can be used during the computation[4].

Moreover, there are no explicit direct means of communication: any communication occurs in a totally implicit manner. Specifically, it happens by means of observing the robots' positions in the plane, and taking a deterministic decision accordingly. In other words, the only mean for a robot to send information to some other robot is to move and let the others observe (reminiscent of bees in a bee dance).

In the following, we will discuss in detail the implications of time settings.

2.2 Activation Schedules

Before proceeding to prove the main result of this paper, we need to describe in more detail the critical feature that regards the way the robots act during

[4] Note that the non obliviousness feature does not imply the possibility for a robot to find out which robot corresponds to which position it stored, since the robots are anonymous.

the computation; that is, the timing of the operations executed by each robot during its life.

In particular, in the model described so far, the amount of time spent in observation[5], in computation, in movement, and in inaction is finite but otherwise unpredictable; then, we say that the robots are *fully asynchronous*. In particular, the robots do not (need to) have a common notion of time. Each robot executes its actions at unpredictable time instants. This setting is adopted in CORDA. If the robots move according to this time setting, we say that they move according to an *asynchronous activation schedule*. Furthermore,

Definition 1. *An algorithm* $\mathcal{A}$ solves *a problem* $\mathcal{P}$ *in* CORDA *if, by activating the robots according to* any *asynchronous activation schedule, the robots reach a configuration such that the task defined by* $\mathcal{P}$ *is accomplished.*

In contrast, if the robots execute their activities (observation, computation, movement, and waiting) in an atomic and instantaneous fashion (that is, the amount spent in each activity of each cycle is negligible), we say that the robots are *atomically synchronized*, and that they move according to an *atomic activation schedule*. This temporal setting was first introduced by Suzuki *et al.* [10]; we will refer to this setting as ATOM.

Let us denote by $\mathfrak{C}$ and $\mathfrak{Z}$ the class of problems that are solvable in the asynchronous and the atomic setting, respectively. The relationship between these two classes is expressed from the following

Theorem 1 ([9]). $\mathfrak{C} \subset \mathfrak{Z}$.

Therefore, in order to prove the impossibility of GATHERING, it is sufficient to show that the problem is unsolvable in the atomic setting.

In an atomic activation schedule, at each time instant t, every robot r_i is either *active* or *inactive*. At least one robot is active at every time instant, and every robot becomes active at infinitely many unpredictable time instants[6]. For any $t \geq 0$, if r_i is inactive, then $p_i(t+1) = p_i(t)$; otherwise $p_i(t+1) = p$, where $p_i(t)$ denotes the position of robot r_i at time instant t, and p is the point returned by $\mathcal{A}$ [10].

Thus, an active robot r_i executes its cycle *atomically* and *instantaneously*, in the sense that a robot that is active and observes at t, has already reached its destination point p at $t+1$, and no fellow robot can see it *while* it is moving (or, alternatively, the movement is *instantaneous*).

We now introduce two general properties that follow from the ATOM setting, and that are not specific to the GATHERING. The first one stresses out the fact that, if a set of robots that at a given time instant t lie on the same position of the plane are all active at time t, then they will behave like they were one robot.

[5] i.e., activating the sensors and receiving their data.

[6] A special case is when every robot is active at every time instant; in this case we say that the robots are *strongly synchronized*. In [2, 10], the authors refer to this case simply as *synchronous*.

Lemma 1. *Let $\mathbb{H}$ be a set of black robots that at time t lie all on the same point $p^t_{\mathbb{H}}$. If all robots in $\mathbb{H}$ are active at time t, then at time $t+1$ all robots in $\mathbb{H}$ will again lie on the same position (possibly different from $p^t_{\mathbb{H}}$).*

Proof. The lemma follows from the fact that $\mathcal{A}$ is deterministic, the robots cannot detect multiplicity, and that all robots in $\mathbb{H}$ clearly have the same view of the world at t.

The following lemma points out that, if all robots in the system take the decision to move towards a point p at the same time instant t, then, even if a subset of them is blocked, all the others will still move towards p.

Lemma 2. *Let us assume that activating all robots at time t they gather on the same point p at time $t+1$, and let $\mathbb{H}$, with $1 \leq |\mathbb{H}| < n$, be any subset of robots that are not on p at t. If all robots not in $\mathbb{H}$ were still activated at t, and all robots in $\mathbb{H}$ were inactive at t, then all r_i, $r_i \notin \mathbb{H}$, will be on p at $t+1$, and all robots in $\mathbb{H}$ will not.*

Proof. The lemma follows from the lack of multiplicity detection and from the fact that $\mathcal{A}$ is deterministic.

3 Is GATHERING Possible?

To our knowledge, in all solutions proposed to solve the GATHERING, the ability of the robots to detect multiplicity (i.e., if on a given point there is more than one robot) is used either implicitly (like in [10]) or explicitly (like in [4]). Moreover, as already mentioned, the only attempt to avoid use of multiplicity detection to solve the problem, produced a solution that works only for non oblivious robots [3]. In other words, all previous solutions make some extra assumption on the capabilities of the robots. In this section, we indeed prove that GATHERING is impossible in general.

In particular, we first focus on ATOM; by Theorem 1, the result extends to CORDA. In the following we assume that the n robots in the system execute only deterministic and oblivious algorithms according to atomic activation schedules. Moreover, we assume $n \geq 3$. In fact, in [10] it has been proven that there exists no oblivious algorithm that solves the problem in a model based on ATOM when $n = 2$, under the assumption that two robots never collide (since they are modeled as no-dimensional points). Therefore, by Theorem 1, this case is unsolvable in CORDA too[7].

Moreover, we denote by $\mathcal{A}$ a generic deterministic and oblivious algorithm, and by $\mathcal{A}_g$ an oblivious deterministic algorithm that correctly solves the gathering problem in ATOM. Recall that $\mathcal{A}_g$ solves the gathering problem if, starting

[7] In [5], however, has been proved that the problem is trivially solvable in CORDA, hence in ATOM, if the robots can collide: in this case, in fact, it is sufficient to move the robots against each other until they gather.

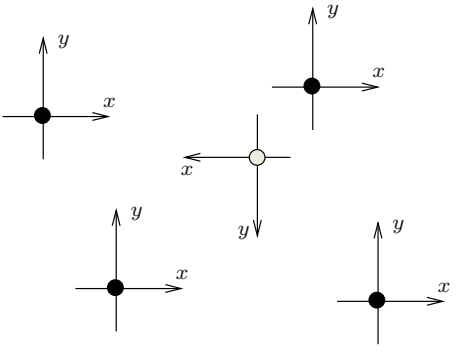


Fig. 1. Orientation of the axes of the black robots and of the white robot, in Assum3

from any valid initial configuration, it lets the robots gather on the same point $\mathfrak{p}$ in finite time: here, a valid initial configuration is a configuration where no two robots occupy the same position on the plane.

Finally, let $\mathbb{H}$ be a set of robots that at time t lie all together on the same point on the plane: in the following, we indicate such a position by $p^t_{\mathbb{H}}$, and by $|\mathbb{H}|$ the number of robots in $\mathbb{H}$.

3.1 The Proof: General Idea

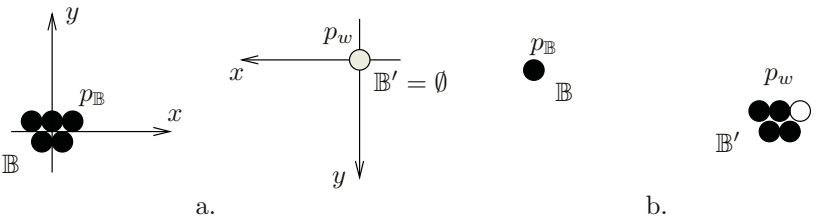
The general idea to prove impossibility of GATHERING is as follows. First, we define a scenario that we will use to defeat any possible $\mathcal{A}_g$. In particular, in this scenario

Assum1. all robots have the same unit distance;

Assum2. $\delta = \delta_1 = \ldots = \delta_n$ (with δ_i as defined in Assumption A1 of Section 2.1);

Assum3. robots $r_1, \ldots, r_{n-1}$, from now on the *black* robots, have the same orientation and direction of the local coordinate system, while r_n, from now on the *white* robot, has a local coordinate system where both axes have the same direction but opposite orientation with respect to the coordinate system of the black robots (see Figure 1). In the following, we denote by p^t_w the position of the white robot at time t. The black and white coloring is used only for the sake of presentation, and this information is not used by the robots during the computation. The same applies for the indices given to the robots (they are anonymous).

We want to stress out, however, that Assum1–Assum3 are not known to the robots; hence they cannot use these information in their computations. Moreover, $\mathcal{A}_g$ correctly solves GATHERING iff the robots gather in finite time regardless their local unit measures, and the local orientation of their axes; hence, $\mathcal{A}_g$ must work also in a scenario described by Assum1–Assum3.



a. b.

Fig. 2. In (a) a $\mathbb{E}_1$-configuration is depicted, while in (b) a $\mathbb{E}_2$-configuration. By Assum3 and since the robots cannot detect multiplicity, in both configurations (and in general in any $\mathbb{E}$-configuration) the white robot has the *same* view of the world as the robots in $\mathbb{B}$. In fact, both r_n and the robots in $\mathbb{B}$ see only one other robot on the point of coordinate (z, z'), with respect to their local coordinate systems

Second, we indeed show that there exists no $\mathcal{A}_g$ that can be executed in such a scenario according to an atomic activation schedule and that allows the robots to gather in a point in finite time. More specifically, we first show that, given $\mathcal{A}_g$, there exists always an atomic activation schedule that brings the robots, in a finite number of cycles, in a particular configuration, called $\mathbb{E}$-configuration, and defined as follows.

Definition 2 ($\mathbb{E}$-configuration). *An $\mathbb{E}$-configuration is a configuration of the robots where (i) the black robots are partitioned in two groups $\mathbb{B}$ and $\mathbb{B}'$, with $\mathbb{B}'$ possibly empty; (ii) the robots in $\mathbb{B}'$ and the white robot r_n lie on the same position p_w, and (iii) the robots in $\mathbb{B}$ lie on a position $p_{\mathbb{B}} \neq p_w$. Moreover, $\mathbb{E}_1$-configuration (shortly $\mathbb{E}_1$) is the $\mathbb{E}$-configuration where $\mathbb{B}' = \emptyset$ (see Figure 2.a), and $\mathbb{E}_2$-configuration (shortly $\mathbb{E}_2$) is the $\mathbb{E}$-configuration where $|\mathbb{B}| = 1$ and $|\mathbb{B}'| = n - 2$ (see Figure 2.b).*

Then, we prove that there exists an atomic activation schedule for $\mathcal{A}_g$ that, starting from a $\mathbb{E}$-configuration, lets the robots loop between $\mathbb{E}$-configurations, always avoiding the gathering.

Assume for a moment that at a given time t robots are in a $\mathbb{E}$-configuration; furthermore, let the robots in $\mathbb{B}$ (resp. the white robot) be active at t, and the robots in $\mathbb{B}'$ inactive for all $t' \geq t$. Then, since the robots cannot detect multiplicity, the robots in $\mathbb{B}$ and the white robot have the same view of the world at time t. Hence, since $\mathcal{A}_g$ is deterministic, we have that

Lemma 3. *If no robot changes position at time $t + 1$, then no robot will ever move, independently from their activation sequences (given that the robots in $\mathbb{B}'$ stay inactive).*

3.2 The Proof

As already outlined in Section 3.1, we first show that a $\mathbb{E}$-configuration can be reached by executing $\mathcal{A}_g$ according to a specific atomic activation schedule,

	t_s	t_s+1	$\cdots$	$t_{\mathbb{E}}-1$	$t_{\mathbb{E}}$
r_1	A	A	$\cdots$	A	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
r_k	A	A	$\cdots$	I	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
r_n	A	A	$\cdots$	A	

Fig. 3. The synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}}$ described in Lemma 4

say $Sync\mathcal{F}_{\mathbb{E}}$. Such a schedule is built as follows: at each cycle, if the robots, all activated, do not compute all the same destination point (according to the definition of $\mathcal{A}_g$), then they are activated and moved towards the destination point they compute. Otherwise, one of them, say r_k, is kept inactive, while all others are activated. In this way, the $n-1$ robots that are active will gather on the same point $\tilde{p}$, while r_k does not; hence, the robots are in a $\mathbb{E}$-configuration. More formally,

Lemma 4. *Given $\mathcal{A}_g$, there exists an atomic activation schedule $Sync\mathcal{F}_{\mathbb{E}}$ for $\mathcal{A}_g$, and a time $t_{\mathbb{E}} > 0$ such that, if the robots do not all occupy the same position on the plane when the execution of $\mathcal{A}_g$ starts, the robots are in $\mathbb{E}_1$ or $\mathbb{E}_2$ at time $t_{\mathbb{E}}$, if the computation is done according to $Sync\mathcal{F}_{\mathbb{E}}$.*

Proof. Let t_s be the time when the computation starts, and $pos_1, \ldots, pos_n$ be the positions occupied by the robots at this time. By hypothesis, there exist at least two positions pos_i and pos_j, $i \neq j$, such that $pos_i \neq pos_j$. $Sync\mathcal{F}_{\mathbb{E}}$ is reported in Schedule 1 (refer to Figure 3 for a pictorial representation).

Schedule 1 Build$_{\mathbb{E}}(t_s, pos_1, \ldots, pos_n)$.

Init. At the beginning, all robots are inactive. Set $t = t_s$, and go to Rule1.

Rule1. If normally activating all robots at time t they are not on the same point $\tilde{p}$ at time $t+1$, then in $Sync\mathcal{F}_{\mathbb{E}}$ all r_i are active at t. Set $t = t+1$, and go to Rule1. Otherwise,

Rule2. let r_k be a robot that is not on $\tilde{p}$ at time t. Then, in $Sync\mathcal{F}_{\mathbb{E}}$ all r_i, $i \neq k$, are active at t, while r_k is inactive at t.

In the following we will show that, starting the execution of $\mathcal{A}_g$ at time t_s according to $Sync\mathcal{F}_{\mathbb{E}}$, all robots are in a $\mathbb{E}_1$-configuration or $\mathbb{E}_2$-configuration at time $t_{\mathbb{E}} > t_s$. In fact, since by hypothesis $\mathcal{A}_g$ solves the problem, after finite time Rule2. is executed; hence $t_{\mathbb{E}}$ is finite. Moreover, until $t_{\mathbb{E}} - 1$ all robots are always active, and at this time, r_k is the only robot to be inactive.

By construction, $t_{\mathbb{E}}$ is the first time such that, if all the robots were normally activated at time $t_{\mathbb{E}} - 1$, they would be on the same position $\tilde{p}$ at time $t_{\mathbb{E}}$. Therefore, since there exists at least two positions pos_i and pos_j at time t_s such

that $pos_i \neq pos_j$, there must exist at least one robot r_k that is not on $\tilde{p}$ at time $t_{\mathbb{E}} - 1$. According to Rule2., r_k is inactive at $t_{\mathbb{E}} - 1$. By Lemma 2, at time $t_{\mathbb{E}}$ all robots r_i, $i \neq k$, are on $\tilde{p}$, and r_k is on a position different from $\tilde{p}$, and the lemma follows.

In the following two lemmas, we show that there is no algorithm that, starting from $\mathbb{E}_1$ or $\mathbb{E}_2$, lets the robots gather in a point.

Lemma 5. *There exists no deterministic oblivious algorithm that, starting from a $\mathbb{E}_1$-configuration, solves the gathering problem in a finite number of cycles for a set of $n \geq 3$ robots that can not detect multiplicity.*

Proof. By contradiction, let $\mathcal{A}_g$ be a deterministic oblivious algorithm that, starting from a $\mathbb{E}_1$-configuration, lets the robots gather in a point in finite time when they cannot detect multiplicity. In the following, we will describe an atomic activation schedule $Sync\mathcal{F}_{\mathbb{E}_1}$ for $\mathcal{A}_g$ such that, if the robots are in a $\mathbb{E}_1$-configuration at a given time t_s and the computation is done according to $Sync\mathcal{F}_{\mathbb{E}_1}$, the robots never gather in the same point $\mathfrak{p}$.

Schedule 2 Build$_{\mathbb{E}_1}(t_s, pos_1, \ldots, pos_n)$.

Init. At the beginning, all robots are inactive. Set $t = t_s$, and go to RuleB1.

RuleB1. If activating one of the black robots at time t, it is not on p_w^t at time $t+1$, then in $Sync\mathcal{F}_{\mathbb{E}_1}$ all black robots are activated at t and moved to the destination point they compute. The white robot is inactive at t. Set $t = t+1$, and go to RuleW1.

RuleB2. Otherwise,

- RuleB2.1 In $Sync\mathcal{F}_{\mathbb{E}_1}$, the black robots $r_1, \ldots, r_{n-2}$ are active at t and moved to the destination point they compute. The black robot r_{n-1} and the white robot r_n are inactive at t. Set $t = t+1$.
- RuleB2.2 In $Sync\mathcal{F}_{\mathbb{E}_1}$, the white robot is active at t and moved to the destination point it computes. All black robots are inactive at t. Set $t = t+1$.
- RuleB2.3 In $Sync\mathcal{F}_{\mathbb{E}_1}$, the black robot r_{n-1} is active at t and moved to the destination point it computes. The black robots $r_1, \ldots, r_{n-2}$ and the white robot r_n are inactive at t. Set $t = t+1$, and go to RuleW1.

RuleW1. If activating the white robot at time t, it is not on $p_{\mathbb{B}}^t$ at time $t+1$, then in $Sync\mathcal{F}_{\mathbb{E}_1}$ the white robot is activated at t and moved to the destination point it computes. The black robots are inactive at t. Set $t = t+1$, and go to RuleB1.

RuleW2. Otherwise,

- RuleW2.1 As in RuleB2.1.
- RuleW2.2 As in RuleB2.2.
- RuleW2.3 As in RuleB2.3, except that at the end of this step go to RuleB1.

Proof. Let $pos_1 = \ldots = pos_{n-1} = p_{\mathbb{B}}^{t_s}$, and $pos_n = p_w^{t_s}$. $Sync\mathcal{F}_{\mathbb{E}_1}$ is reported in Schedule 2 (refer to Figure 5 for a pictorial representation).

It follows from the definition of $\mathbb{E}_1$ that, at t_s, $p_{\mathbb{B}}^{t_s} \neq p_w^{t_s}$. $Sync\mathcal{F}_{\mathbb{E}_1}$ moves alternatively the black robots (as a group) and the white robot, until at time

t either the black robots compute as destination point p_w^t, or the white robot computes as destination point $p_{\mathbb{B}}^t$. When this happens, the gathering is avoided

1. by first moving all the black robots but one on p_w^t; then,
2. by moving the white robot on $p_{\mathbb{B}}^t$; and finally,
3. by moving the last black robot (still on $p_{\mathbb{B}}^t$) on p_w^t;

that is the black robots and the white robot are forced to switch their positions.

First note that, after every execution of RuleB1. all black robots *must change* position, and move *all together* towards the new destination (different from the position occupied by the white robot). In fact, let $t^* \geq t_s$ be a time instant when RuleB1. starts being executed, and such that all robots in $\mathbb{B}$ are on the same position $p_{\mathbb{B}}^{t^*} \neq p_w^{t^*}$. It follows from the definition of $Sync\mathcal{F}_{\mathbb{E}_1}$ that at t^* all black robots are active. If all these robots are still on $p_{\mathbb{B}}^{t^*}$ at the end of RuleB1. (that is at time $t^* + 1$), then by Lemma 3 no robot would ever move, hence the robots would never gather on the same point. Therefore, the robots in $\mathbb{B}$ cannot be on $p_{\mathbb{B}}^{t^*}$ at the end of RuleB1., and they *must change* position. Furthermore, since there is a black robot that, if active at t^*, would reach a position $p \neq p_w^{t^*}$ at time $t^* + 1$, by Lemma 1 the black robots will reach *all together* p at time t^*+1, with $p \neq p_w^{t^*}$ and $p \neq p_{\mathbb{B}}^{t^*}$. Symmetrically, it follows that, if RuleW1. starts at time t^*, the white robot will be on a position $p \neq p_w^{t^*}$ and $p \neq p_{\mathbb{B}}^{t^*}$ at time $t^* + 1$, while all black robots are inactive at t^* (hence they are still on $p_{\mathbb{B}}^{t^*}$ at time $t^* + 1$). Therefore, as long as RuleB1. or RuleW1. are executed, the robots are in $\mathbb{E}_1$-configurations.

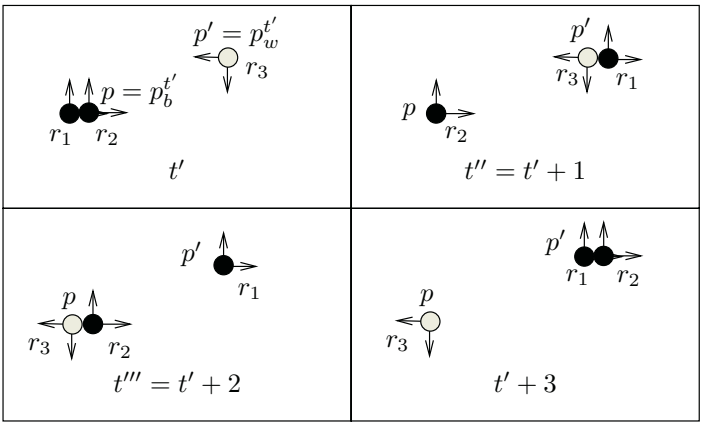


Fig. 4. Execution of RuleB2. in schedule $\mathbf{Build}_{\mathbb{E}_1}()$ in Lemma 5, with $n = 3$. At time t' each robot sees only one other robot; in particular, r_1 and r_2 see one robot on the point of coordinate (z, z') (with respect to their local coordinate system), and r_3 sees one robot on the point of coordinate (z, z') (with respect to its local coordinate system). That is, all the robots have the *same* view of the world. This view of the world is observed also by r_3 at time t'', and by r_2 at time t'''

	t_s	B1.	W1.		B2.1	B2.2	B2.3	W1.
r_1		A	I	$\cdots$	A	I	I	$\cdots$
$\vdots$		$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$
r_{n-2}		A	I	$\cdots$	A	I	I	$\cdots$
r_{n-1}		A	I	$\cdots$	I	I	A	$\cdots$
r_n		I	A	$\cdots$	I	A	I	$\cdots$

Fig. 5. The synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}_1}$ described in Lemma 5. Here is depicted the case when RuleB2. is invoked first

Since, by hypothesis, $\mathcal{A}_g$ solves the problem, after a finite number of cycles either RuleB2. or RuleW2. is executed. Without loss of generality, let us assume that RuleB2. is executed first, say at time $t' > t_s$ (the case when RuleW2. is executed first can be handled similarly). Thus, according to $Sync\mathcal{F}_{\mathbb{E}_1}$, $n-2$ black robots are active at time t', while r_{n-1} and r_n are inactive (RuleB2.1). This rule is chosen because there is a black robot that, if normally activated at t', would compute $p_w^{t'}$ as destination point. Hence, by Lemma 1, the $n-2$ active robots will leave $p = p_{\mathbb{B}}^{t'}$ and reach $p' = p_w^{t'}$ (Figure 4).

At this point, RuleB2.2 is invoked at time $t'' = t' + 1$: the white robot is active at t'', while all black robots are inactive. By Assum1–Assum3 and since multiplicity cannot be detected, r_n has the same view of the world that the black robots that moved in RuleB2.1 had at time t' (refer to Figure 4); specifically, the white robot sees only one robot, that is the last black robot r_{n-1} that at this time is still on p (r_{n-1} is inactive at t' and t''). As a consequence, since $\mathcal{A}_g$ is oblivious and deterministic, the result of the *Compute* state of r_n at t'' is the same as the result of the *Compute* state that the black robots performed at time t' (in RuleB2.1): that is, r_n decides to reach the only other robot it sees (r_{n-1}), hence r_n computes p as destination point. Therefore, at time $t'' + 1$ the white robot reaches r_{n-1} on p.

Finally, RuleB2.3 is started at time $t''' = t'' + 1$: the last black robot r_{n-1} (still on p) is active at t''', while all the other black robots (at this time on p') and r_n (on p) are inactive. At time t''', r_{n-1} has the same view of the world that the black robots that moved in RuleB2.1 had at time t'; specifically, since it can not distinguish multiplicity, it sees all other black robots (on p') as one robot. Therefore it computes p' as destination point, and reaches all the other black robots at time $t''' + 1$.

In conclusion, if RuleB2.1 is started at time t', at time $t'''+1 = t'+3$ all black robots are on p', and the white robot is on p. That is, the black and white robots simply switched positions, and at time $t'+3$ they are again in a $\mathbb{E}_1$-configuration. Therefore, by executing $\mathcal{A}_g$ according to $Sync\mathcal{F}_{\mathbb{E}_1}$, the robots will never gather on the same point. This leads to a contradiction, and the lemma follows.

Lemma 6. *In* Corda *there exists no deterministic oblivious algorithm that, starting from a $\mathbb{E}_2$-configuration, solves the gathering problem in a finite number of cycles for a set of $n \geq 3$ robots that can not detect multiplicity.*

Proof. By contradiction, let $\mathcal{A}_g$ be a deterministic oblivious algorithm that, starting from a $\mathbb{E}_2$-configuration, lets the robots gather in a point in finite time when they cannot detect multiplicity. Similarly to the previous lemma, we will describe a synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}_2}$ for $\mathcal{A}_g$ such that, if the robots are at a given time t_s in a $\mathbb{E}_2$-configuration and the computation is done according to $Sync\mathcal{F}_{\mathbb{E}_2}$, the robots never gather in the same point $\mathfrak{p}$. By Lemma 1,

Schedule 3 Build$_{\mathbb{E}_2}(t_s, pos_1, \ldots, pos_n)$.

Init. At the beginning, all robots are inactive. Set $t = t_s$, and go to Rule1.

Rule1. If activating all robots at time t, they are not on the same position $\tilde{p}$ at time $t+1$, then in $Sync\mathcal{F}_{\mathbb{E}_2}$ all robots are normally activated. Set $t = t+1$, and go to Rule1.

Rule2. Otherwise,

- Rule2.1 If no robot is on $\tilde{p}$ at time t, then in $Sync\mathcal{F}_{\mathbb{E}_2}$ all robots in $\mathbb{B}'$ and r_{n-1} are active at t and moved to the destination point they compute. The white robot r_n is inactive at t. Set $t = t+1$, and go to RuleB1. defined in Lemma 5.
- Rule2.2 If r_n is on $\tilde{p}$ at time t, then all robots in $\mathbb{B}'$ are active at t, while r_{n-1} and r_n are inactive at t. Set $t = t+1$, and go to Rule1.
- Rule2.3 If r_{n-1} is on $\tilde{p}$ at time t, then all robots in $\mathbb{B}'$ are active at t, while r_n and r_{n-1} are inactive at t. Set $t = t+1$, and go to RuleB1. in Schedule 2.
- Rule2.4 If all robots in $\mathbb{B}'$ are on $\tilde{p}$ at time t, then r_{n-1} is active at t, while the robots in $\mathbb{B}'$ and r_n are inactive. Set $t = t+1$, and go to RuleB1. in Schedule 2.

as long as Rule1. is executed, all robots in $\mathbb{B}'$ move always all together; hence, at any time, they always occupy the same position on the plane. Since by hypothesis $\mathcal{A}_g$ solves the problem, after a finite number of cycles Rule2. is executed, say at time t', and let $\tilde{p}$ as defined in Rule2., that is the point where the robots would gather if all active at t'.

It follows from the definition of $\mathbb{E}_2$ that at the beginning $p_{\mathbb{B}}^{t_s} \neq p_w^{t_s}$. Without loss of generality, let us assume that $r_1, \ldots, r_{n-2}$ are the black robots in $\mathbb{B}'$ (at t_s they lie on $p_w^{t_s}$), and that r_{n-1} is the only robot in $\mathbb{B}$.

$Sync\mathcal{F}_{\mathbb{E}_2}$ moves all robots until they decide to gather on the same point (eventually this happens, since by hypothesis $\mathcal{A}_g$ solves the problem); in particular, all robots in $\mathbb{B}'$ are forced to move together, hence to lie always on the same point. When this happens, the robots are forced to reach either a $\mathbb{E}_1$ or a $\mathbb{E}_2$-configuration. At this point, $Sync\mathcal{F}_{\mathbb{E}_2}$ behaves exactly like $Sync\mathcal{F}_{\mathbb{E}_1}$ described in the previous lemma; hence it avoids the gathering. Let $pos_1, \ldots, pos_{n-2}, pos_n = p_w^{t_s}$, and $pos_{n-1} = p_{\mathbb{B}}^{t_s}$. $Sync\mathcal{F}_{\mathbb{E}_2}$ is reported in Schedule 3 (refer to Figure 6 for a pictorial representation).

First, note that it is impossible that at time t' the robots in $\mathbb{B}'$ and r_n are already on $\tilde{p}$, while the only robot in $\mathbb{B}$ is not. In fact, let us assume that r_n and the robots in $\mathbb{B}'$ are already on $\tilde{p}$ at time t'; thus, the robots are in a $\mathbb{E}$-configuration at t'. Rule2. is executed at t' because, if all the robots were active at t', they would

be on $\tilde{p}$ at time $t' + 1$; hence, since by hypothesis r_n and the robots in $\mathbb{B}'$ are already on $\tilde{p}$ at time t', these robots would not move between time t' and $t' + 1$. Therefore, is like the robots in $\mathbb{B}'$ are inactive at t'. Hence, by Lemma 3, no robot would change position between time t' and $t' + 1$, hence they would not gather on $\tilde{p}$ at time $t' + 1$, and Rule2. would not have been executed at time t'. Similarly, it can be proven that

> it is impossible that at time t' the robot in $\mathbb{B}$ and r_n are already on $\tilde{p}$, while the robots in $\mathbb{B}'$ are not (it is sufficient to switch the roles of $\mathbb{B}$ and $\mathbb{B}'$ in Lemma 3); and
>
> it is impossible that at time t' the robot in $\mathbb{B}$ and those in $\mathbb{B}'$ are already on $\tilde{p}$, while r_n is not.

Moreover, since by hypothesis t' is the first time such that activating all robots, they would gather on the same point, it can not be that all robots are already on $\tilde{p}$ at t'. In the following, we analyze the remaining possible cases.

1. *No robot is on $\tilde{p}$ at time t'.* In this case, Rule2.1 is executed, and r_n is inactive at t'. Hence, by Lemma 2, at time $t' + 1$ all robots but r_n are on $\tilde{p}$; that is, the robots are in a $\mathbb{E}_1$-configuration.
2. *Only r_n is already on $\tilde{p}$ at time t'.* In this case, Rule2.2 is executed, and the robots in $\mathbb{B}'$ are active at t', while r_{n-1} and r_n are inactive. Hence, by Lemma 2, at time $t' + 1$ all robots in $\mathbb{B}'$ and r_n are on $\tilde{p}$, while r_{n-1} is not. That is, the robots do not gather in $\tilde{p}$ at $t' + 1$, and they are again in a $\mathbb{E}_2$-configuration.
3. *Only r_{n-1} is already on $\tilde{p}$ at time t'.* In this case, Rule2.3 is executed: at t', r_{n-1} and r_n are inactive, while the robots in $\mathbb{B}'$ are active. By Lemma 2, at time $t' + 1$ all robots but r_n are on $\tilde{p}$; that is, the robots are in a $\mathbb{E}_1$-configuration.
4. *Only the robots in $\mathbb{B}'$ are already on $\tilde{p}$ at time t'.* Rule2.4 is executed. Using an argument similar to the one used in the previous case, it follows that also in this case the robots are in a $\mathbb{E}_1$-configuration at time $t' + 1$.

In conclusion, at time $t' + 1$, either the robots are in a $\mathbb{E}_1$-configuration or again in a $\mathbb{E}_2$-configuration. In the first case, the lemma follows by Lemma 5. In the second case, either Rule2.2 is never executed again after $t' + 1$, or every time it is executed the robots are once again in a $\mathbb{E}_2$-configuration. In both cases, the lemma follows.

To summarize, thus far we proved that,

> given any algorithm $\mathcal{A}_g$, there exists an atomic activation schedule that, starting from any valid configuration for the gathering problem, brings the robots either in a $\mathbb{E}_1$ or $\mathbb{E}_2$-configuration in a finite number of cycles (Schedule 1);

	t_s Rule1.	Rule1.		Rule2.	B1.
r_1	A	A	$\cdots$	A	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$
r_{n-2}	A	A	$\cdots$	A	$\cdots$
r_{n-1}	A	A	$\cdots$	A	$\cdots$
r_n	A	A	$\cdots$	I	$\cdots$

Fig. 6. The synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}_2}$ described in Lemma 6. The case when Rule2.1 is executed first is depicted

there exists no deterministic oblivious algorithm that, starting form a $\mathbb{E}_1$ or $\mathbb{E}_2$-configuration, solves the gathering problem in a finite number of cycles (Schedules 2 and 3 in the Appendix).

Hence, by Lemmas 4–6, and by Theorem 1, it follows that

Theorem 2. *In* CORDA *and* ATOM*, there exists no deterministic oblivious algorithm that solves the* GATHERING *problem in a finite number of cycles, hence in finite time, for a set of* $n \geq 2$ *robots.*

References

1. N. Agmon and D. Peleg. Fault Tolerant Gathering Algorithms for Autonomous Mobile Robots. In *Proc.* 15^{th} *Symposium on Discrete Algorithms (SODA 2004)*, pages 1063–1071, 2004.
2. H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.
3. M. Cieliebak. Gathering non-oblivious mobile robots. In *Proc. Latin American Conf. on Theoretical Informatics (LATIN '04)*, LNCS 2976, pages 577–588, 2004.
4. M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving The Gathering Problem. In *Proc.* 30^{th} *International Colloquium on Automata, Languages and Programming (ICALP '03)*, pages 1181–1196, 2003.
5. M. Cieliebak and G. Prencipe. Gathering Autonomous Mobile Robots. In *Proc. of* 9^{th} *International Colloquium On Structural Information And Communication Complexity (SIROCCO 9)*, pages 57–72, June 2002.
6. R. Cohen and D. Peleg. Robot Convergence via Center-of-gravity Algorithms. In *Proc.* 11^{th} *International Colloquium On Structural Information And Communication Complexity (SIROCCO 11)*, pages 79–88, 2004.
7. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In *Proc.* 10^{th} *Annual International Symposium on Algorithms and Computation (ISAAC '99)*, LNCS 1741, pages 93–102, December 1999.
8. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Autonomous Mobile Robots With Limited Visibility. In *Proc.* 18^{th} *International Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, LNCS 2010, pages 247–258, February 2001.

9. G. Prencipe. The Effect of Synchronicity on the Behavior of Autonomous Mobile Robots. *Theory of Computing Systems*, 2004. (accepted for publication).
10. I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam Journal of Computing*, 28(4):1347–1363, 1999.

Majority and Unanimity in Synchronous Networks with Ubiquitous Dynamic Faults

Nicola Santoro[1] and Peter Widmayer[2]

[1] School of Computer Science, Carleton University, Canada
santoro@scs.carleton.ca
[2] Institut for Theoretical Informatics, ETH Zurich, Switzerland
widmayer@inf.ethz.ch

Abstract. In this paper we are interested in synchronous distributed systems subject to transient and ubiquitous failures. This includes systems where failures will occur on *any* communication link, systems where *every* processor will fail at one time or another, etc., and, following a failure, normal functioning can resume after a finite (although unpredictable) amount of time. Notice that these cases cannot be handled by the traditional *component failure* models.

The model we use is the *transmission failure* model, known also as the *dynamic faults* model. Using this model, we study the fundamental problem of *agreement* in synchronous systems of arbitrary topology.
We establish bounds on the number of dynamic faults that make any non-trivial form of agreement (even strong majority) impossible; in turn, these bounds express connectivity requirements which must be met to achieve any meaningful form of agreement. We also provide, constructively, bounds on the number of dynamic faults in spite of which any non-trivial form of agreement (even unanimity) is possible.

These bounds are shown to be tight for a large class of networks, that includes hypercubes, toruses, rings, and complete graphs; incidentally, we close the existing gap between possibility and impossibility of non-trivial agreement in complete graphs in presence of dynamic Byzantine faults.

None of these results is derivable in the component failure models; in particular, all our *possibility* results hold in situations for which those models indicate *impossibility*.

1 Introduction

1.1 The Framework

In this paper, we are concerned with the fundamental problem of *agreement* in synchronous systems where failures have mostly a transient and ubiquitous nature; that is, faults can occur anywhere in the system and, following a failure, normal functioning can resume after a finite (although unpredictable) time.

The reality of these systems is not fully captured by the *component failure* models proposed in the literature.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 262–276, 2005.

Consider, for instance, the *processor failure* model (e.g., see [13, 14, 16, 18, 25]). In this model, only processors can be faulty; any other type of failures is inscribed to the faulty behaviour of some of the involved processors: if a message is lost, either the sending or the receiving processor will be declared faulty; once this happens, that processor will be forever considered faulty. This leads to undesirable conclusions: in the case of ubiquitous failures where any processor may occasionally lose messages (a situation that clearly occurs in real systems), the entire system will be declared unusable for any computation, even if the system is synchronous.

Analogous undesirable situations occur in the *link failure* model and in the *hybrid failure* models that consider both links and processors (e.g. [6, 28, 31]). Some attempts have been made to remedy this situation by modifying the model allowing some failed processors and links to recover (e.g., [1, 22]), but they are just partial and limited to eventually-synchronous systems.

In this paper we are interested in synchronous distributed systems subject to (possibly transient) ubiquitous failures. This includes systems where failures will occur on any communication link, systems where every processor will experience at one time or another send or receive failure, etc. Notice that these cases can not be handled by the traditional component failure models.

The model we use is the *transmission failure* model, known also as the *dynamic faults* model, introduced in [29] and investigated e.g. in [5, 7, 8, 9, 10, 11, 12, 19, 20, 24, 26, 30]. In this model, a *transmission* is a pair (α, β) of messages $\alpha, \beta \in M \cup \{\Omega\}$ for a pair (i, j) of neighbouring processors called *source* and *destination*, where M is a fixed and possibly infinite message universe and Ω is the null message: α is the message sent by the source and β is the message received by the destination; by convention $\alpha = \Omega$ denotes that no message is sent, and $\beta = \Omega$ denotes that no message is received. A transmission (α, β) is *faulty* if $\alpha \neq \beta$, non-faulty otherwise. In this model, the only failures occurring in the system are transmission faults, and failures are fully *dynamic*: the set of pairs $(source, destination)$ whose transmissions are faulty may change at every clock cycle. An instance of this model is the *single mobile failure* described in [27].

Notice that component failures can be easily modeled by transmission faults; for instance, loss (i.e., $\alpha \neq \Omega = \beta$) of all messages sent by and to a processor can be used to describe a crash failure of that processor; analogously, it is easy to describe all the other subtypes of faults considered in the component failure model: send and receive failures, Byzantine failures, etc. Similarly, the crash failure of a link can be modeled by the loss of all messages sent between the two incident processors. In other words, the processor and the link failure models can be seen as a special *static* cases of the transmission failure model, where all the faults are restricted to the transmissions involving a fixed (though, a priori unknown) set of processors or of links.

In this paper, we are concerned with the fundamental problem of *agreement* in presence of *dynamic* transmission faults in synchronous systems of *arbitrary* topology.

1.2 Related Work

Synchronous agreement in the component failure models is perhaps the most intensively and extensively studied problem in distributed computing. We will just note that most of the work has focused on the *complete graph* (e.g., [14, 16, 17, 23, 25, 31]); fewer studies have focused on other classes of graphs (e.g., [4, 15, 17, 25]) or on arbitrary networks [13].

In the dynamic faults model, the studies on agreement have focused on synchronous systems whose communication topology is the *complete graph*, and both possibility and impossibility results have been established [29, 30]. In particular, for these systems the established bounds have been shown to be tight in the case of *omission* (i.e., $\alpha \neq \beta = \Omega$) failures, as well as in the case of any mix of *corruption* (i.e., $\Omega \neq \alpha \neq \beta \neq \Omega$) and *addition* (i.e., $\Omega = \alpha \neq \beta$) failures. In the case of *Byzantine* (i.e., arbitrary) transmission failures there was however a gap between the possibility and impossibility bounds.

The link between conditions for (partial) *broadcast* and for possibility of (partial) agreement with dynamic faults was established in [29]. Most of the subsequent research on dynamic faults has focused on reliable broadcast in the case of *omission* failures; the problem has been investigated in complete graphs [10, 26, 30], hypercubes [8, 11, 19], tori [7, 12], star graphs [8], as well as in arbitrary topologies [5]. The more general problem of evaluation of Boolean functions has been studied only for complete networks [9, 30]; computation of some special functions has been investigated also in the case of *anonymous* networks [9].

1.3 Main Contributions

In this paper we present several results, extending the existing knowledge on agreement with dynamic faults to topologies other than the complete network.

Impossibility
First of all, we prove a general theorem characterizing classes of faulty transmissions for which any non-trivial agreement is *impossible*. As a corollary, we prove several impossibility results, some of them quite unexpected, and clearly not inferable from the existing results of the other models.

Consider for example a d-dimensional *hypercube* and let us consider the occurrence of just d transmission faults per clock cycle. With d *omissions* per clock cycle, we can simulate the crash failure of a single processor; hence, *unanimity* clearly can not be expected. What about other levels of agreement ?

Something can be learned from the processor failure model. For the hypercube, Dolev's results in the processor failure model [13] state that unanimity among the non-faulty processors is possible if the number of the faulty processors is less than d. If the d omissions per clock cycle are localized to a single processor, there is only one "faulty" processor; hence, by Dolev's result, an agreement among $n - 1$ processors is possible.

What happens if those d omissions per clock cycle are *dynamic* (i.e., not localized to a single processor) ? Since with d omissions per clock cycle at most

a single processor can be isolated from the rest, one might still reasonably expect that an agreement among $n-1$ processors can be reached even if the faults are dynamic. Not only this expectation is false, but we prove that any form of non-trivial agreement can not be reached under those conditions. In fact, even strong majority is impossible.

If the transmission faults are arbitrary (omissions, additions, and corruptions: the *Byzantine* case), the gap between the static and dynamic cases is even stronger. By Dolev's result [13], it follows again that in the hypercube an *agreement among* $n-1$ processors is possible in spite of d Byzantine transmission faults if they are statically restricted to the messages sent by a single processor. From our results with omissions, we already know that if the faults are dynamic then strong majority is impossible; if the faults are Byzantine, we show that this is so even if the number of faults is just $\lceil d/2 \rceil$.

These results for the hypercube are instances of the more general results obtained here. We consider *arbitrary networks* and establish bounds on the number of transmission faults that make any non-trivial form of agreement impossible; in turn, these bounds express connectivity requirements that must be met to achieve any meaningful form of agreement. Let $G = (V, E)$ be the network topology, and let d be its max degree. We prove that:

1. With d omissions per clock cycle, strong majority cannot be reached.
2. If the failures are any mixture of corruptions and additions, the same bound d holds for the impossibility of strong majority.
3. In the case of arbitrary faults (omissions, additions, and corruptions: the Byzantine case), strong majority cannot be reached if just $\lceil d/2 \rceil$ transmissions may be faulty.

Figure 1 summarizes these findings.

These results are established using the proof structure for dynamic faults introduced in [29]. Although based on the bivalency argument of Fischer, Lynch

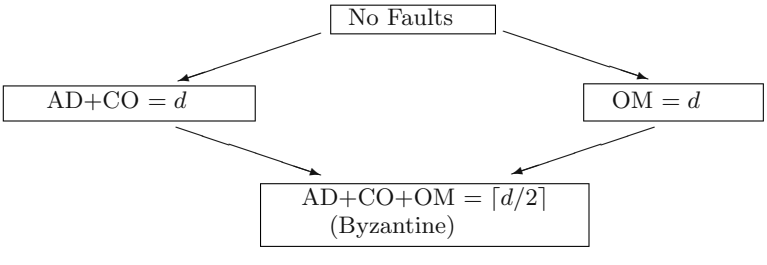


Fig. 1. Impossibility. Minimum number of faults per clock cycle that may render **strong majority** impossible

and Paterson [18], the framework differs significantly from the ones for asynchronous systems since we are dealing with a *fully synchronous* system[1] where time is a direct computational element, with all its consequences; e.g., the clock value is explicitly part of the state of a processor; non-delivery of an expected message is detectable, unlike asynchronous systems where a "slow" message is indistinguishable from an omission; etc.

Possibility
We then turn to the possibility of agreement in spite of dynamic faults. We examine all the combinations of different types of faults, and for each we establish bounds for achieving *unanimity* among the processors. The results we obtain vary with the types of faults and are sometimes counterintuitive.

We first consider the case when the faults are just *omissions*. It is known that, in the d-dimensional *hypercube*, if the faults are at most $d-1$ omissions per clock cycle, then broadcast (and thus unanimity) is possible (e.g., [8, 11]). These results are actually just instances of the more general results established here.

In fact, we prove that, in any network G, if the faults are *omissions*, then *unanimity* can be reached if the number of faults per clock cycle is at most $c-1$, where c is the connectivity of G.

Interestingly, if the faults are *corruptions*, we show that unanimity can always be achieved regardless of the number of faults. This is true also in systems where the faults are only *additions*. On the other hand, the combination of *additions and corruptions* creates a $c-1$ threshold for unanimity.

In the more complex *Byzantine* case of *omissions, additions and corruptions*, we prove that unanimity is still possible if at most $\lceil c/2 \rceil - 1$ transmissions per clock cycle may be faulty. A summary of all the possibility results is shown in Figure 2.

Let us stress that, regardless of any analogy with bounds established in the component failure models, none of these results is implied or derivable from those models. On the contrary, these *possibility* results are obtained with a number and type of faults for which all the component failure models indicate *impossibility*.

Tightness
For all systems, except those where faults are just corruptions or just additions (and in which unanimity is possible regardless of faults), the bounds we have established are similar except that the possibility ones are expressed in terms of the *connectivity* c of the graph, while the impossibility ones are in terms of the *degree* d of the graph.

This means that, in the case of *d-connected graphs*, the impossibility bounds are indeed tight:

[1] A bivalency approach for impossibility results in synchronous systems, first used in [29], has been recently employed also in [2, 3, 27].

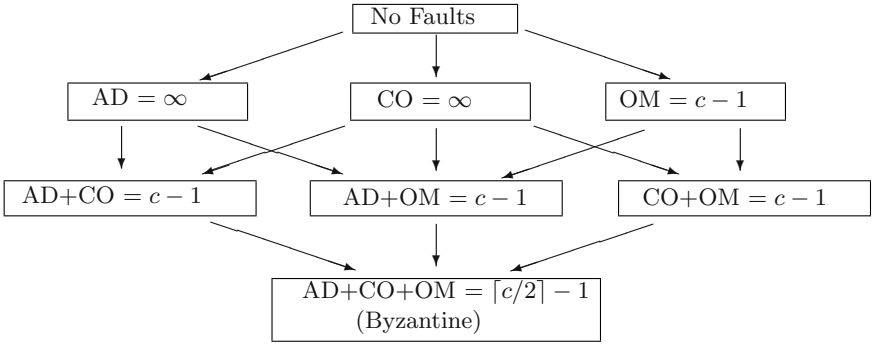


Fig. 2. Possibility. Maximum number of faults per clock cycle in spite of which **unanimity** is possible

1. with the number of faults (or more) specified by the impossibility bound, even strong majority is impossible;
2. with one less fault than specified by the impossibility bound, even unanimity can be reached, and
3. any agreement among less than a strong majority of the processors can be reached without any communication.

In other words, in these systems, agreement is either trivial or complete or impossible.

This large class of networks includes hypercubes, toruses, rings, complete graphs, etc. As a consequence, we also close the existing gap in [29, 30] between possibility and impossibility for non-trivial agreement with dynamic Byzantine faults in complete graphs.

For those graphs where $c < d$ there is a gap between possibility and impossibility. Closing this gap is clearly a goal of future research.

2 Definitions and Terminology

Most of the terminology is taken from [29]; some of it was in turn a modification and adaptation for the synchronous case of that of [18].

A k-agreement protocol $\mathcal{P}$ is a synchronous system of $n \geq 2$ processors $p_1, \ldots, p_n$ connected through dedicated communication links. The connection structure is an arbitrary undirected, connected simple graph $G = (V, E)$. An edge joining two nodes of the graph is a bidirectional link between the two processors. The degree of a node p_i in the graph is denoted by d_i; d is the maximum node degree. The graph G is said to be a *d-edge-connected graph* if, between any

two nodes, there exist d edge-disjoint paths, (one might say that neighbors are d-connected). Rings, toruses, hypercubes and complete graphs are examples of d-edge-connected graphs.

Processors communicate by sending messages to adjacent processors. The system operates in complete *synchrony*: each processor has direct read-only access to a global clock, and every message sent at time t is received (if no error occurs) and processed at its destination at time $t+1$. The message sent by some processor need not be the same for all destination processors, i.e., separate links allow for different messages to different destinations at the same time. Each processor p_i has a message register holding a message vector $m_i \in (M \cup \{\Omega\})^{d_i}$, where M is a fixed and possibly infinite message universe, and Ω is the null element indicating absence of communication. The component m_{ij} of m_i is the message to be sent from p_i to its neighbor p_{i_j}.

A *transmission* is a pair (α, β) of messages; α is the sent message and β is the received message; by convention, $\alpha = \Omega$ denotes that no message is transmitted, and $\beta = \Omega$ denotes that no message is received. Let Φ denote the set of all transmissions.

A transmission (α, β) is *faulty* if $\alpha \neq \beta$, *non-faulty* otherwise. Faulty transmissions can be partitioned into three sets, corresponding to the three types of faults considered in this paper:

- *omissions*: $\mathcal{O} = \{(\alpha, \beta) \in \Phi : \alpha \neq \Omega = \beta\}$
  (a sent message is never delivered to a destination processor);

- *additions*: $\mathcal{A} = \{(\alpha, \beta) \in \Phi : \alpha = \Omega \neq \beta\}$
  (a message is delivered to a processor, although no message was sent);

- *corruptions*: $\mathcal{C} = \{(\alpha, \beta) \in \Phi : \Omega \neq \alpha \neq \beta \neq \Omega\}$
  (a sent message is delivered with different content to a destination processor).

Each processor p_i has a one-bit input register[2] with an initial value $x_i \in \{0, 1\}$, an output register for which it must choose a value $v_i \in \{0, 1\}$ as the result of its computation, and an unbounded amount of local storage. The values of the registers and of the global clock, together with the program counters and the internal storage, comprise the *internal state* of each processor.

For each processor, the *initial state* prescribes fixed starting values for all but the input register; in particular, the output register starts with null value Ω, and the clock starts with value 0. Each processor acts deterministically; it can never change the input register nor the clock, and can change the value of its output register only once, from Ω to $v \in \{0, 1\}$. The states in which the output register has value $v \in \{0, 1\}$ are distinguished as being *v-decision-states*.

[2] For simplicity, we describe here Boolean agreement; all the results hold also for non-binary values.

The system reaches a *k-agreement* if within finite time k processors choose the same value $v \in \{0,1\}$ subject to the condition that, if all values x_i were the same, then v must be that value. Note that if $k \leq \lceil n/2 \rceil$, a k-agreement can be trivially reached without any communication (e.g., each p_i chooses x_i for v). In this paper, we are interested in *non-trivial* agreements (i.e., $k > \lceil n/2 \rceil$), especially $(\lceil n/2 \rceil + 1)$-agreement (*strong majority*) and n-agreement (*unanimity*).

A *message matrix* A$= (\alpha_{ij})$ is a $n \times n$–matrix of messages, where $\alpha_{ij} := \delta$ if $(p_i, p_j) \notin E$, for a special $\delta \in M$ not used otherwise; by definition, $(p_i, p_i) \notin E$.

A *transmission pattern* τ for A is a set of $n(n-1)$ transmissions $\tau[i,j]$, $i \neq j$, of the form (α, β), where $\alpha = \alpha_{ij}$ and $\tau[i,j] = (\delta, \delta)$ iff $(p_i, p_j) \notin E$. $\tau[i,j] = (\alpha, \beta)$ indicates that p_i sends α to p_j and, as a result of this transmission, p_j receives β from p_i. The message system maintains a non-empty set T(A) of transmission patterns for each message matrix A. It supports the abstract operation *transmit*(A): return a transmission pattern $\tau \in$ T(A). Thus, if $|\mathrm{T(A)}| > 1$, the message system acts non-deterministically.

A *configuration* of the system consists of the internal state of each processor. An *initial* configuration is one in which all processors are in an initial state.

A *step* takes one configuration C to another and consists of the following atomic (indivisible) sequence of operations, where $\lambda(C)$ is the message matrix defined by the contents of the message registers in C:

1. a *transmit*$(\lambda(C))$ is performed to obtain a transmission pattern τ for $\lambda(C)$;
2. the global clock is incremented by one unit;
3. depending on p_i's internal state, on the current clock value and on the received messages (as specified by τ), each processor p_i enters a new internal state.

Since processors act deterministically, a step is completely determined by the transmission pattern τ; τ is called an *event*, and $\tau(C)$ denotes the resulting configuration. For $t > 0$, let $R^t(C) = R(R^{t-1}(C))$, where $R^0(C) = R(C) = \{\tau(C) : \tau \in \mathrm{T}(\lambda(C))\}$ is the set of all possible configurations resulting from C in one step. The configurations in $R(C)$ are sometimes called *succeeding configurations of C*. Let $R^*(C) = \{C' : \exists t \geq 0,\ C' \in R^t(C)\}$ be the set of configurations reachable from C. A configuration that is reachable from some initial configuration is said to be *accessible*.

Let $v \in \{0,1\}$. A configuration C has *decision value* v if at least k processors are in a v-decision state; note that if $k > \lceil n/2 \rceil$, a configuration can have at most one decision value. A configuration C is *v-valent* if there exists a $t \geq 0$ such that all $C' \in R^t(C)$ have decision value v; that is, a v-valent configuration will always result in at least k processors deciding on v. A configuration C is *bivalent* if there exist in $R^*(C)$ both a 0-valent and a 1-valent configuration.

A set S of transmission patterns (events) is *w-admissible*, $0 \leq w \leq 2|E|$, if

1. for each message matrix A, $\exists \tau \in S$ for A;
2. every $\tau \in S$ contains at most w faulty transmissions;
3. $\exists \tau \in S$ which contains exactly w faulty transmissions.

Given a set of events S, a k–agreement protocol $\mathcal{P}$ *admits* w *faults* in S if S is w-admissible and $\cup_A \mathrm{T(A)} = S$ (i.e., the message system returns only events in S). A k–agreement protocol is *correct* in spite of w transmission faults in S if

1. for each $v \in \{0, 1\}$, some accessible configuration has decision value v;
2. it admits w faults in S;
3. for each initial configuration C there exists a $t \geq 0$ such that every $C' \in R^t(C)$ has a decision value.

3 On Agreement in Spite of Faults

In this section we generalize to arbitrary networks the theorems of [29] for complete graphs.

Let $s_i(C)$ denote the internal state of p_i in C. From the definitions of state and of event, the next two properties follow immediately.

Property 1. For two configurations C' and C'', let $\lambda(C') = (\alpha_{ij})$ and $\lambda(C'') = (\beta_{ij})$ be the corresponding message matrices. If $s_j(C') = s_j(C'')$ for some processor p_j, then $(\alpha_{j1}, ..., \alpha_{jn}) = (\beta_{j1}, ..., \beta_{jn})$.

Property 2. Let C' and C'' be two configurations such that $s_j(C') = s_j(C'')$ for some processor p_j, and let τ' and τ'' be events for $\lambda(C')$ and $\lambda(C'')$, respectively. Let $\tau'[i, j] = (\alpha'_{i,j}, \beta'_{i,j})$ and $\tau''[i, j] = (\alpha''_{i,j}, \beta''_{i,j})$. If $\beta'_{i,j} = \beta''_{i,j}$ for all i, then $s_j(\tau'(C')) = s_j(\tau''(C''))$.

Two configurations C' and C'' are *l-adjacent* if $s_i(C') = s_i(C'')$ for all $i \neq l$; they are *adjacent* if they are l-adjacent for some l.

A set S of events is *l-adjacency-preserving* if for any two l-adjacent configurations C' and C'' there exist in S two events τ' and τ'' for $\lambda(C')$ and $\lambda(C'')$, respectively, such that $\tau'(C')$ and $\tau''(C'')$ are l-adjacent. S is *adjacency-preserving* if is l-adjacency-preserving for all l.

A set S of events is *continuous* if for any configuration C and for any $\tau', \tau'' \in S$ for $\lambda(C)$, there exists a finite sequence $\tau_0, \ldots, \tau_m$ of events in S for $\lambda(C)$ such that $\tau_0 = \tau', \tau_m = \tau''$, and $\tau_i(C)$ and $\tau_{i+1}(C)$ are adjacent, $0 \leq i < m$.

Lemma 1. *If S contains all possible events with at most w faults, then S is continuous.*

Note that, in Lemma 1, it is not necessary, but sufficient for S to contain *all possible* events with at most w faults.

Theorem 1. *Let S be continuous, k-adjacency-preserving and w-admissible, $w > 0$. Let $\mathcal{P}$ be a $(\lfloor (n-1)/2 \rfloor + 2)$–agreement protocol. If $\mathcal{P}$ contains two accessible k-adjacent configurations, a 0-valent and a 1-valent one, then $\mathcal{P}$ is not correct in spite of w transmission faults in S.*

Theorem 2. *Let S be adjacency-preserving, continuous and w-admissible. Then no k-agreement protocol is correct in spite of w transmission faults in S for $k > \lceil n/2 \rceil$.*

Proof. Assume $\mathcal{P}$ is a correct $(\lceil n/2 \rceil + 1)$-agreement protocol in spite of w transmission faults when the message system returns only events in S. In a typical bivalency approach, the proof involves two steps: first it is argued that there is some initial configuration in which the decision is not already predetermined; second, it is shown that it is possible to forever postpone entering a configuration with a decision value.

Lemma 2. *$\mathcal{P}$ has an initial bivalent configuration.*

Lemma 3. *A bivalent configuration has a succeeding bivalent configuration.*

From Lemmas 2 and 3 it follows that there exists an infinite sequence of accessible bivalent configurations, each derivable in one step from the preceding one. This contradicts the assumption that for each initial configuration C there exists a $t \geq 0$ such that every $C' \in R^t(C)$ has a decision value; thus, $\mathcal{P}$ is not correct. This concludes the proof of Theorem 2.

The above theorems provide a powerful tool for proving impossibility results for non-trivial agreement: if it can be shown that a set S of events is adjacency-preserving, continuous, and w-admissible, then by Theorems 1 and 2 no non-trivial agreement is possible for the types and numbers of faults implied by S. Obviously, not every set S of events is adjacency-preserving.

4 Impossibility of Strong Majority

4.1 Omission Faults

We show that no *strong majority* protocol is correct in spite of d transmission faults, even when the faults are only omissions. For message matrix $\mathrm{A} = (\alpha_{ij})$, let $\mathrm{O(A)}$ be the set of all events τ on A defined as follows: for at most d pairs $(p_i, p_j) \in E, \tau[i,j] = (\alpha_{ij}, \Omega)$, and for all other pairs $(p_i, p_j) \in E, \tau[i,j] = (\alpha_{ij}, \alpha_{ij})$. Then, $\mathbf{O} := \bigcup_A \mathrm{O(A)}$ is the set of all events containing at most d omission faults.

Lemma 4. **O** *is d–admissible, continuous and adjacency-preserving.*

Then, by Theorem 2, we immediately get

Theorem 3. *No k-agreement protocol $\mathcal{P}$ is correct in spite of d transmission faults in* **O** *for $k > \lceil n/2 \rceil$.*

4.2 Addition and Corruption Faults

Here, we show that no strong majority protocol is correct in spite of d transmission faults, when the faults are additions and corruptions. For message matrix

A = (α_{ij}), let AC(A) be the set of all events τ on A defined as follows: for at most d pairs $(p_i, p_j) \in E, \tau[i,j] = (\alpha_{ij}, \beta_{ij}), \alpha_{ij} \neq \beta_{ij}$, where $\beta_{ij} \neq \Omega$ if $\alpha_{ij} \neq \Omega$, and for all other pairs $(p_i, p_j) \in E$, $\tau[i,j] = (\alpha_{ij}, \alpha_{ij})$. Then $\mathbf{AC} := \bigcup_A$ AC(A) is the set of all events containing at most d addition and corruption faults.

Lemma 5. AC *is d–admissible, continuous and adjacency-preserving.*

Then, by Theorem 2, we immediately get

Theorem 4. *No k-agreement protocol $\mathcal{P}$ is correct in spite of d transmission faults for* **AC** *for $k > \lceil n/2 \rceil$.*

4.3 Byzantine Faults

We show that no strong majority protocol is correct in spite of $\lceil d/2 \rceil$ arbitrary transmission faults. For a message matrix A = (α_{ij}), let ACO(A) be the set of all events τ on A containing at most $\lceil d/2 \rceil$ faulty transmissions. Then $\mathbf{ACO} := \bigcup_A$ ACO(A) is the set of all events containing at most $\lceil d/2 \rceil$ transmission faults, where the faults may be omissions, corruptions and additions.

Lemma 6. ACO *is $\lceil d/2 \rceil$–admissible, continuous and adjacency-preserving.*

Theorem 5. *No k–agreement protocol P is correct in spite of $\lceil d/2 \rceil$ transmission faults in* **ACO** *for $k > \lceil n/2 \rceil$.*

5 Possibility Results

Let c be the (edge-)connectivity of G; then there are at least c edge-disjoint paths between any pair of nodes. This fact has been used in the component failure model to show that, with enough redundant transmissions, information can be correctly propagated in spite of faults and that the processors can reach some form of agreement (e.g. [13, 15]). Our results on the possibility of agreement in spite of a certain amount w of *dynamic* faults also exploit this fact; the value of w obviously depends on the nature of the faults.

Our general strategy is to first prove that it is possible to correctly broadcast the value of a bit within a fixed amount of time T in spite of w faults per clock cycle. This reliable broadcast, once established, can then be trivially used to correctly compute the logical OR of the input values, enabling the processors to reach *unanimity* in spite of those w faults per clock cycle. The reliable broadcast protocols will differ depending on the nature of the faults.

5.1 Single Type Faults

It is well known that it is possible to broadcast in spite of $w \leq c - 1$ *omissions* per clock cycle; let $T^*(G)$ denote the time this process requires in G. Hence

Theorem 6. *Let the system faults be* omissions. *Unanimity can be reached in spite of $w = c - 1$ faults per clock cycle in time $T = T^*(G)$.*

Currently, the best available bound is $T^*(G) = O(diam(G)^{w+1})$ [5], where $diam(G)$ is the diameter of the graph. Better bounds are known for specific networks [7, 8, 11, 12, 19, 26, 30]; interestingly, in a hypercubes, $diam(G) + 2$ clock cycles are known to suffice [19].

Surprisingly, if the faults are just *corruptions*, unanimity can be reached regardless of the number of faults.

Theorem 7. *Let the system faults be* corruptions. *Unanimity can be reached regardless of the number of faults in time* $T = diam(G)$.

In case of just *additions*, if all processors transmit to their neighbours in each clock cycle, they avoid the creation of unintended messages. Thus, they can correctly compute the OR using a simple diffusion mechanism: initially, a processor sends its value; if at any time it is aware of the existence of a 1 in the system, it will only send 1 from that moment on. The process clearly terminates after $diam(G)$ clock cycles. Hence

Theorem 8. *Let the system faults be* additions. *Unanimity can be reached regardless of the number of faults in time* $T = diam(G)$.

5.2 Composite Faults

If the system suffers of *omissions and corruptions* or *omissions and additions*, the situation is fortunately no worse than that of systems with only omissions.

Theorem 9. *Unanimity can be reached in spite of* $w = c-1$ *faults per clock cycle if the system faults are* omissions and corruptions*, or* omissions and additions. *In both systems, the time to agreement is* $T = T^*(G)$.

For systems with the two other types of composite dynamic faults, we achieve the *reliable broadcast* of 1 using a more complex mechanism for *reliable transmission* of a bit to a neighbour.

Consider a processor p_i and a neighbour p_j; let $D_{i,j}$ be the set of the c (shortest) disjoint paths from p_i to p_j, including the direct link (p_i, p_j). Let $l_{i,j}$ be the max length of any one of those paths, and let $l = \max_{i,j}\{l_{i,j}\}$. To communicate a bit to p_j, p_i will send it along all the c paths in $D_{i,j}$ for a number t of communication cycles, where t depends on the type of faults, and will be discussed later. Every processor p on one of those paths, upon receiving the message for p_j, will forward it only along the path. Note that incorrect path information (due to corruptions and/or additions) in a message for p_j received by x is *detectable* since (1) x knows the neighbour z from which it received the message; and (2) x can determine if z is its predecessor in the claimed path to p_j. A message with incorrect path information will be discarded. The specifics of the communication process depend on the faults in the system.

In the case of both *additions and corruptions*, using the technique of [30], we distinguish between *even* and *odd* clock cycles; an even clock cycle and

its successive odd cycle constitute a *communication cycle*. Processor p_i will transmit its value only during t successive *even* (resp. *odd*) cycles if the bit is 0 (resp. 1). Processors on the path, receiving a bit during an even (resp. odd) cycle, will forward it in the next even (resp. odd) cycles, regardless of its value.

In case of *Byzantine* faults, a communication cycle lasts only one clock cycle; that is, any received message is forwarded along the path immediately.

In both cases,

Lemma 7. *In* absence of faults, p_j *will receive at least* $(l-1)+c(t-(l-1))$ *copies of the message from* p_i *within* t *communication cycles.*

Lemma 8. *If* w *is the maximum number of faults per clock cycle, then in* t *communication cycles at most* $w\ t$ *copies of the message from* p_i *are lost or arrive incorrectly at* p_j*.*

In case of *additions and corruptions*, p_j can determine the correct content of the bit in spite of receiving (possibly conflicting) values both in even and odd cycles.

Lemma 9. Reliable transmission *is possible in spite of* $c-1$ *additions and corruptions and uses* $2(c-1)(l-1)$ *clock cycles.*

Hence, reliable broadcast can be guaranteed if at most $c-1$ additions and corruptions occur in the system:

Theorem 10. *Let the system faults be* additions and corruptions. *Unanimity can be reached in spite of* $w = c-1$ *faults per clock cycle; the time is* $T \leq 2\ diam(G)\ (c-1)\ (l-1)$.

In case of *Byzantine* faults, the decision process (i.e., how p_j, out of the possibly conflicting received messages, determines the correct content of the bit) is simple: p_j selects as correct the bit value received most often during the t time units. In this case, the decision is correct if the number of correct copies of the message received by p_j is greater than the number of faulty. By Lemmas 7 and 8, for this to happen it must be $w \leq \lceil d/2 \rceil - 1$ and thus $t \geq (c-1)(l-1)$.

Lemma 10. Reliable transmission *tolerates* $\lceil c/2 \rceil - 1$ *Byzantine faults per clock cycle, and uses* $(c-1)(l-1)$ *clock cycles.*

Hence, reliable broadcast can occur in spite of $\lceil c/2 \rceil - 1$ Byzantine faults.

Theorem 11. *Let the system faults be* arbitrary. *Unanimity can be reached in spite of* $w = \lceil c/2 \rceil - 1$ *faults per clock cycle; the time is at most* $T \leq diam(G)\ (c-1)\ (l-1)$.

5.3 Tightness of Impossibility Bounds

The results of Section 4 and Section 5 together show that the established *impossibility* bounds for agreement protocols are indeed *tight* in the case of *d-connected graphs*:

1. with the number of faults (or more) specified by the impossibility bound, even *strong majority* is impossible;
2. with *one less fault* than specified by the impossibility bound, even *unanimity* can be reached, and
3. any agreement among *less than a strong majority* of the processors can be reached *without any communication*.

This large class of networks includes hypercubes, toruses, rings, complete graphs, etc. As a consequence, we also close the existing gap in [29, 30] between possibility and impossibility for non-trivial agreement with dynamic Byzantine faults in *complete graphs*.

For those graphs where $c < d$, the results established here leave a gap between possibility and impossibility. Closing this gap is the goal of future investigations.

References

1. M.K. Aguilera, W. Chen, and S. Toueg, "Failure detection and consensus in the crash-recovery model". *Distributed Computing* 13 (2), 99-125, 2000.
2. M.K. Aguilera and S. Toueg, "A simple bivalency proof that t-resilient consensus requires t+1 rounds". *Information Processing Letters* 71, 155-158, 1999.
3. Z. Bar-Joseph and M. Ben-Or, "A tight lower bound for randomized synchronous consensus", *Proc. ACM Symp. on Principles of Distributed Computing* (PODC 98), Puerto Vallarta, 193-199, 1998.
4. M. Ben-Or and D. Ron, "Agreement in presence of faults on networks of bounded degree", *Information Processing Letters* 57 (6), 329 - 334, 1996.
5. B.S. Chlebus, K. Diks, and A. Pelc, "Broadcasting in synchronous networks with dynamic faults". *Networks* 27, 309-318, 1996.
6. F. Cristian, H. Aghili, R. Strong, and D. Dolev, "Atomic broadcast: From simple message diffusion to Byzantine agreement". *Information and Computation* 118 (1), 158 - 179, 1995.
7. G. De Marco and A. Rescigno, "Tighter bounds on broadcasting in torus networks in presence of dynamic faults". *Parallel Processing Letters* 10, 39-49, 2000.
8. G. De Marco and U. Vaccaro, "Broadcasting in hypercubes and star graphs with dynamic faults", *Information Processing Letters* 66, 309-318, 1998.
9. S. Dobrev, "Computing input multiplicity in anonymous synchronous networks with dynamic faults". In *Proc. 26th Int. Workshop on Graph-Theoretic Concepts in Computer Science* (WG 2000), LNCS 1928, 138-148, 2000.
10. S. Dobrev, "Communication-efficient broadcasting in complete networks with dynamic faults". In *Proc. 9th Coll. on Structural Information and Communication complexity (SIROCCO'02)*, 101-113, 2002.

11. S. Dobrev and I. Vrt'o, "Optimal broadcasting in hypercubes with dynamic faults". *Information Processing Letters* 71, 81-85, 1999.
12. S. Dobrev and I. Vrt'o, "Optimal broadcasting in even tori with dynamic faults". *Parallel Processing Letters* 12, 17-22, 2002.
13. D. Dolev, "The Byzantine generals strike again". *J. Algorithms* 3 (1), 14-30, 1982.
14. D. Dolev and H. R. Strong, "Polynomial algorithms for multiple processor agreement". In *Proc. 14th ACM Symp. on Theory of Computing* (STOC 82), 401-407, 1982.
15. C. Dwork, D. Peleg, N. Pippenger and E. Upfal, "Fault tolerance in networks of bounded degree", *SIAM J. Computing* 17 (5), 975-988, 1988.
16. M. J. Fischer and N.A. Lynch, "A lower bound for the time to assure interactive consistency", *Information Processing Letters* 14 (4), 183-186, 1982.
17. M. J. Fischer, N.A. Lynch, and M. Merritt, "Easy impossibility proofs for distributed consensus problems", Distributed Computing 1 (1) 26-39, 1986.
18. M. J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of distributed consensus with one faulty process", *J. ACM* 32 (2), 1985.
19. P. Fraigniaud and C. Peyrat, "Broadcasting in a hypercube when some calls fail", *Information Processing Letters* 39, 115-119, 1991.
20. L. Gasienic and A. Pelc, "Broadcasting with linearly bounded faults", *Discrete Applied Mathematics* 83, 121-133, 1998.
21. J. Garay and Y. Moses, "Fully polynomial Byzantine agreement for $n > 3t$ processors in $t + 1$ rounds". *SIAM J. Computing* 27 (1), 247-290, 1998.
22. R. Guerraoui and R. R. Levy, "Robust emulation of shared memory in a crash-recovery model. In *Proc. 24th Int. Conf. on Dist. Computing Systems* (ICDCS'04), 400-407, 2004.
23. V. Hadzilacos, "Connectivity requirements for Byzantine agreement under restricted types of failures". *Distributed Computing* 2 , 95-103, 1987.
24. R. Kralovic, R. Kralovic, and P. Ruzicka, "Broadcasting with many faulty links". In *Proc. 10th Coll. on Structural Information and Communication complexity* (SIROCCO'03), 211-222, 2003.
25. L. Lamport, R. Shostak and M. Pease, "The Byzantine generals problem". *ACM Trans. Programming Languages and Systems* 4 (3), 382-401, 1982.
26. Z. Liptak and A. Nickelsen, "Broadcasting in complete networks with dynamic edge faults", In *Proc. 4th Int. Conf. on Principles of Distributed Systems* (OPODIS 00), Paris, 123-142, 2000.
27. Y. Moses and S. Rajsbaum, "A Layered Analysis of Consensus", *SIAM J. Computing* 31 (4), 989 - 1021, 2002.
28. K. J. Perry and S. Toueg, "Distributed agreement in the presence of processor and communication faults". *IEEE Trans. Software Engineering* SE-12 (3), 477-482, March 1986.
29. N. Santoro and P. Widmayer, "Time is not a healer". In *Proc. 6th Symposium on Theoretical Aspects of Computer Science* (STACS 89), 304 - 313, 1989.
30. N. Santoro and P. Widmayer, "Distributed function evaluation in the presence of transmission faults". In *Proc. Int. Symposium on Algorithms* (SIGAL 90), 358 - 367, 1990.
31. U. Schmid and B. Weiss, "Formally verified Byzantine agreement in presence of link faults". In *Proc. 22nd Int. Conf. on Distributed Computing Systems* (ICDCS 02), 608-616, 2002.

Minimizing the Number of ADMs in SONET Rings with Maximum Throughput

Mordechai Shalom and Shmuel Zaks*

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

Abstract. SONET ADMs are dominant cost factors in WDM/SONET rings. Whereas most previous papers on the topic concentrated on the number of wavelengths assigned to a given set of lightpaths, more recent papers argue that the number of ADMs is a more realistic cost measure. The minimization of this cost factor has been investigated in recent years, where single-hop and multi-hop communication models, with arbitrary traffic and uniform traffic loads have been investigated. As a first attempt to understand the trade-off between the number of wavelengths and the number of ADMs, we concentrate on the all-to-all, uniform traffic instance with multi-hop, splittable communication. We look for a solution which makes a full use of the bandwidth and uses the minimum possible number of ADMs under this constraint. We develop an architecture based on successive nested polygons and present a necessary and sufficient condition for a solution in this architecture to be feasible. This architecture leads to a solution using $O(W \log W + N)$ ADMs (compared to NW ADMs for the basic architecture in [1]) which is optimal for $W = O(N/\log N)$. We further improve this result to $O(W \log \overline{W} + N)$ ADMs, where $\overline{W} = o(W)$.

Keywords: Wavelength Assignment, Wavelength Division Multiplexing(WDM), Optical Networks, SONET, Add-Drop Multiplexer(ADM).

1 Introduction

1.1 Background

A single fiber-optic cable offers a bandwidth that can potentially carry information at the rate of several terabits per second, much faster than any electronic device can handle. In order to utilize the potential of optical fiber, wavelength-division multiplexing (WDM) is used. The bandwidth is partitioned into a number of channels at different wavelengths. Several signals can be transmitted through a fiber link simultaneously on different channels. The number of channels (wavelengths) available in WDM systems is limited by the chosen technology.

* This research was supported in part by the fund for the promotion of research at the Technion, by B. and I. Green research fund.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 277–291, 2005.

One of the important parameters affected by the technology is the network cost. Add/drop multiplexers (ADMs) are employed at the network nodes to insert lightwaves into the fiber and extract them.

WDM ring networks are deployed by a growing number of telecom carriers. The problem of minimizing the number of wavelengths has been extensively studied. Variants of this problem such as to maximize the number of lightpaths given a limited number of wavelength (the $MAXPC$ problem) or minimize the blocking probability of a lightpath were also studied.

Among others, in [2] and [3] it is argued that a more realistic cost measure is the number of ADMs used by the network. Moreover, these studies concentrate on a ring topology, since higher level networks which make use of the WDM network may not support arbitrary topologies. The most widely deployed network above the WDM layer is the SONET/SDH self-healing rings, and these networks have to be configured in rings for protection purposes.

The problem of minimizing the additional overhead resulting from the need of these lightpaths to be configured as rings is studied in the literature. This can be split into two problems:

- Assign a route to a lightpath; namely, choose one of two possible directions on the ring such that the maximum number of lightpaths intersecting on an edge is minimal. This is called the ring loading problem. In [4] an optimal solution for the problem in directed rings is given. As for undirected rings, a polynomial time approximation scheme is given in [5].
- Given the routing above, assign wavelengths to the paths such that the number of ADMs used by the system is minimized. We focus on this problem.

The number of ADMs is determined as follows. Each lightpath uses one ADM at each endpoint. An ADM in a common endpoint of two lightpaths can be shared if they are colored with the same color.

A number of previous works [6, 2, 1, 3, 7, 8, 9] studied the minimum ADM problem in which each traffic stream has a predetermined routing. In ring networks this is also called the *arc version* of the problem. The problem is proved to be $\mathcal{NP} - hard$ [3]. For ring networks heuristic algorithms were presented in [2, 3, 10]. Subsequently $3/2, 10/7 + \epsilon$ and $10/7$-approximation algorithms were presented in [7, 8, 9], respectively. The problem is investigated for general topologies in [6].

Generally, the wavelength allocation problems studied so far can be viewed in two categories: The problem of minimizing the number of wavelengths used and the problem of minimizing the number of ADMs used. In [2] it is pointed out that these objective functions may lead to different results; in particular, these objective functions cannot always be minimized simultaneously.

1.2 Our Work

As an attempt to investigate the relationship between the number of wavelengths and the number of ADMs, in this work we concentrate on the uniform all-to-all communication pattern and require maximum utilization of the bandwidth of

W wavelengths. Furthermore we assume the multi-hop communication model and splittable requests, and investigate the problem of minimizing the number of ADMs used under these conditions.

We propose an architecture of successive nested polygons and give a necessary and sufficient condition for a solution in this architecture to be feasible. Using this condition we give an optimal solution for $W = 2$ and a solution using $O(W \log \overline{W} + N)$ ADMs for the general case, where $\overline{W}$ is $o(W)$ and N is the size of the ring. In other words, if $W < N/\log N$ the cost depends asymptotically only on N, and if $W > N^{1+\epsilon}/\log N$ for any $\epsilon > 0$, then the cost depends only on W.

Our technique is extendable to sequences of polygons which are not necessarily nested. The question of the performance of a feasible solution using this architecture, is open.

In Section 2 we give a formal definition of the problem, in Section 3 the basic properties of the solution are investigated. In Section 4 we introduce our demand function which is important for the result. In Section 5 we present the architecture and analyze its performance. In Section 6 we conclude the results and mention further research directions.

2 Problem Definition

Consider W bidirectional SONET rings with (the same) N nodes $\{0, 1, ..., N-1\}$, each operating on a separate wavelength and one ADM for each wavelength at each node. Each ring consists of N lightpaths and traffic can be switched between the rings at each node. This architecture is called PPWDM ([1]).

Consider also the uniform all-to-all traffic where the traffic from node i to node j is

$$\mathcal{T}(i,j) = \begin{cases} 0 \text{ if } i = j \\ \tau \text{ otherwise.} \end{cases}$$

As we will be interested mostly in asymptotic results, and the differences in the results between odd and even values of N are small, we will assume for simplicity that N is even.

Consider the shortest path routing of the above traffic $\mathcal{T}$, where traffic from node i to node $(i + N/2) \mod N$ is split and routed equally on both directions. Figure 1 shows the routing of the demands $\mathcal{T}(0, j)$. The load induced on the system by any node in any direction is

$$\sum_{j=1}^{N/2-1} \tau j + \frac{\tau}{2}\frac{N}{2} = \left(\frac{(N/2-1)N/2}{2} + \frac{N}{4}\right)\tau = \frac{N^2}{8}\tau$$

To obtain the total load induced by all nodes we multiply the above by the number of nodes (N). Since it is clear that the load is the same on each directed edge, we conclude that in this specific routing of the traffic demand $\mathcal{T}$, the load on every directed edge is $\frac{N^2}{8}\tau$.

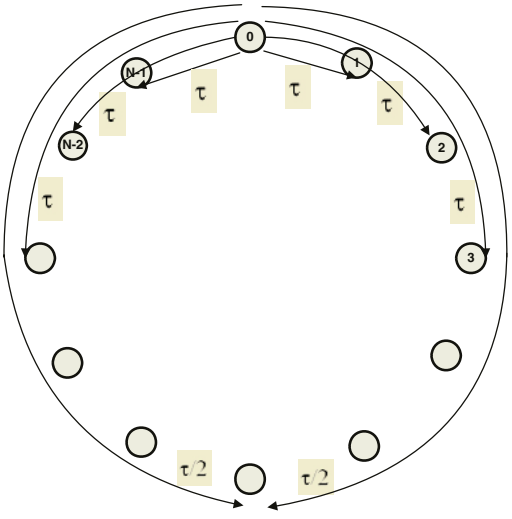


Fig. 1. Routing of traffic from node 0

Clearly the above total load is the minimum possible because of the shortest path routing. Moreover, this total load is distributed evenly on all the edges. Therefore in any routing there is at least one edge with this load or more, in other words this is the minimum possible maximum edge load.

Assuming that the unit of traffic is the capacity of one wavelength, the capacity of each edge is W. The maximum all to all uniform traffic that can be routed on the above PPWDM ring satisfies: $\frac{N^2}{8}\tau = W$, or $\tau = \frac{8W}{N^2}$. For the rest of the paper $n \stackrel{def}{=} \frac{N}{2}$, therefore:

$$\tau = \frac{2W}{n^2}.$$

As traffic can be switched freely between the rings at each node and the capacity of each edge is equal to its load, this traffic can be routed on a PPWDM ring which uses NW ADMs.

Our goal is to find an architecture that uses the same number W of wavelengths, supports the same traffic demand ($\mathcal{T}$) and uses smallest possible number of ADMs.

Proposition 1. *Every solution should use a shortest path routing.*

Otherwise the total load will increase and the average load will be greater than $\frac{n^2}{2}\tau$ $(= W)$. Therefore, there will be at least one edge with load greater than its capacity.

For this reason and the fact that each edge has the same capacity in each direction, the problem can be separated into two identical "directed" problems, one for each direction. We will deal with the "clockwise" problem, in which there

are N directed edges $(i, i+1)$ and the traffic from node i to node j is positive only when the shortest path from i to j is "clockwise".

An architecture is defined by its lightpaths and the routing of the traffic over these lightpaths.

– **Lightpaths:**

Definition 1. *A lightpath is a dipath p of the cycle and a wavelength (color) $w(p) \in \mathbb{N}$ assigned to it.*

Definition 2. *A coloring w is valid if any two lightpaths p and p' such that $w(p) = w(p')$ have no edges in common.*

Definition 3. *A Lightpath Graph is a directed multigraph with N nodes, and an edge $e = (i, j)$ for each lightpath from node i to node j. For such an edge $l(e) \stackrel{def}{=} (j - i) \mod N$ and $w(e)$ is the color assigned to the lightpath it represents.*

The number of ADMs used at each node v of the lightpath graph is the number of colors "touching" v, namely $|\{w(e)|e \text{ is adjacent to } v\}|$.
The number of ADMs used by a lightpath graph is the sum of the number of ADMs used at each node.

– **Routing:** The routing problem is the following multi-commodity flow problem:

- **Input:**
  - A Lightpath Graph and capacities $c(e) = 1$ for all edges.
  - A demand matrix:

$$D(i,j) = \begin{cases} \tau & \text{if } 0 < (j-i) \mod N < N/2 \\ \tau/2 & \text{if } (j-i) \mod N = N/2 \\ 0 & \text{otherwise} \end{cases}$$

  of different commodities.
- **Output:** A flow of the above commodities, completely satisfying the demands.

Our problem is to find a lightpath graph (and a valid coloring of it) with as few ADMs as possible, admitting a routing of the commodities $D(i, j)$.

3 Preliminaries

Proposition 2. *The circle $(0, 1, ..., N-1, 0)$ is a subgraph of the lightpath graph of any solution.*

This is because for each edge $e = (i, i+1)$ the traffic $\mathcal{T}(i, i+1)$ is non-zero and should be routed on the shortest path which is formed by a single lightpath consisting of e only. Therefore each such edge e is an edge of the lightpath graph.

Proposition 3. *The Lightpath Graph of a solution is Circular Eulerian:*

Consider the links e and e' entering and leaving a node. Their capacities are both W and fully used. The capacity dedicated to passthrough traffic is the same in both of them, and uses the same set of wavelengths. Therefore, the capacity dedicated to the remaining traffic is the same and uses the same wavelengths in both edges. Therefore the out degree of any node equals to its in degree. The underlying graph is connected, otherwise there are two distinct nodes i and j such that $D(i,j)$ can not be routed.

As such, this graph can be decomposed into simple cycles, each of which will be called a *polygon*.

Definition 4. *A polygon is a sequence of distinct nodes beginning with the least numbered node. The multiplicity of a polygon is the number of maximal increasing subsequences of this sequence. A polygon with multiplicity 1 is a convex polygon.*

Lemma 1. *Any solution can be decomposed into convex polygons.*

Proof. Any solution is a valid coloring of the lightpaths with W colors. Let E_c be the set of edges in the lightpath graphs such that the corresponding lightpath is colored (assigned wavelength) c. Let: $l(c) = \sum_{e \in E_c} l(e)$. For any color c, $l(c) \leq N$, because otherwise there is at least one edge containing two or more lightpaths with the same color, rendering the coloring invalid. On the other hand $\sum_{e \in E} l(e) = \sum_{c=1}^{W} l(c) = WN$, because there are W edges (lightpaths) using any physical link. By the pigeonhole principle for all c, $l(c) = N$. The lightpaths of E_c do not overlap and the sum of their lengths is N, therefore they form a convex polygon. □

Corollary 1. *The number of ADMs used by a color c is the number $|E_c|$ of the edges of the corresponding convex polygon.*

In view of the preceding results our design problem can be formulated as follows: Find W polygons with minimum total number of edges (nodes) such that routing problem has a solution.

4 The Demand Function

In this section we introduce the demand function which is important in our analysis:

Definition 5. *Given a demand matrix d and an edge e of a Lightpath Graph, we define:*

$$d(e) \stackrel{def}{=} \sum_{i,j} d(i,j)$$

where the sum is taken over all the node pairs i, j such that e is in the direction of the path (on the directed circle) from i to j.

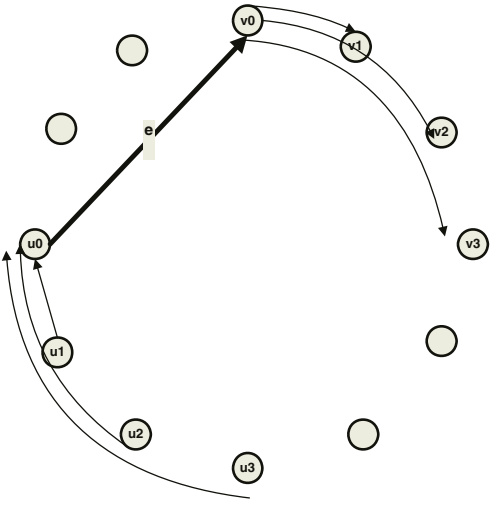


Fig. 2. Demands routable on e

In other words, $d(e)$ is the total demand that can potentially be routed on the edge e.

For the demand matrix D in Section 2, we define similarly $D(e) = \sum_{i,j} D(i,j)$.

In Figure 2 we present a network with $N = 12$ nodes. For the edge e depicted in the figure, we have $D(e) = D(u_3, v_0) + D(u_2, v_0) + D(u_2, v_1) + D(u_1, v_0) + D(u_1, v_1) + D(u_1, v_2) + D(u_0, v_0) + D(u_0, v_1) + D(u_0, v_2) + D(u_0, v_3)$. Note that the summation includes the pairs (u_i, v_j) for which the shortest path from u_i to v_j include $e = (u_0, v_0)$.

Definition 6. *Given a polygon P, we define:*

$$d(P) \stackrel{def}{=} \min \{d(e) | e \in P\}$$

Proposition 4. *In every solution, all the edges of the lightpath graph satisfy $D(e) \geq 1$.*

Otherwise there is an edge with unused capacity under any shortest path routing. But the demand matrix can be routed only by using the full capacity of all the edges.

Corollary 2. *In every solution, all the polygons of the lightpath graph satisfy $D(P) \geq 1$.*

Lemma 2. *Given a lightpath graph, we have:*

$$D(e) > \widetilde{D}(l(e)) \stackrel{def}{=} W \left(1 - \frac{l(e)}{n}\right)^2$$

for every edge e.

Proof. Consider an edge $e = (a, b)$ with $l(e) = l$ and a pair of nodes u and v such that e is on the shortest path from u to v. Let $i = a - u$ and $j = v - b$. Clearly $v - u \leq n$. But $v - u = (v - b) + (b - a) + (a - u) = j + l + i$, therefore $i + j \leq n - l$. The pairs of nodes satisfying this condition contribute τ to $D(e)$ except one pair satisfying $v - u = n$ which contributes $\tau/2$. We get:

$$\begin{aligned} D(e) &= \sum_{\{(i,j)|i+j<n-l\}} \tau + \sum_{\{(i,j)|i+j=n-l\}} \frac{\tau}{2} \\ &= \tau \left(\sum_{s=1}^{n-l} s + \frac{1}{2}(n-l+1) \right) \\ &= \tau \left[\frac{(n-l)(n-l+1)}{2} + \frac{n-l+1}{2} \right] \\ &= \frac{W}{n^2}(n-l+1)^2 = W \left(1 - \frac{l}{n}\right)^2 + O(\frac{1}{n}) \end{aligned}$$

□

Note that $D(e)$ depends only in the length $l(e)$ of e. With some abuse of notation, we will use $D(l(e))$ and $D(e)$ with the same meaning. We will use the following simple properties of the demand function:

$$\lim_{n\to\infty}(D(e) - \widetilde{D}(l(e))) = 0$$
$$D^{-1}(x) \geq \widetilde{D}^{-1}(x) = \left(1 - \frac{\sqrt{x}}{\sqrt{W}}\right) n$$

$\widetilde{D}$ and $\widetilde{D}^{-1}$ are both decreasing functions.

5 Nested Polygons

5.1 Definitions

Definition 7. *Consider two edges $e = (i, j)$ and $e' = (i', j')$ of a lightpath graph. Assume w.l.o.g. that $i = 0$. The edges are said to be:*

$$\begin{array}{ll} \textit{disjoint} & \textit{if } \ j \leq i' < j' \\ \textit{crossing} & \textit{if } \ i' < j < j' \\ \textit{contained} & \textit{if } \ i' < j' \leq j \end{array}$$

In the last case e' is said to be contained in e.

Definition 8. *A convex polygon P' is nested in another convex polygon P, if P is a cyclic permutation of some subsequence of P'.*

Proposition 5. *If a polygon P' is nested in polygon P, then any pair of edges $e \in P$ and $e' \in P'$, are either disjoint or e' is contained in e.*

Definition 9. *A sequence $P_1, P_2, ..., P_k$ of polygons is a nested sequence of polygons if for all $i < k$, P_{i+1} is nested in P_i.*

5.2 Properties of Nested Polygons

Lemma 3. *If a nested sequence of polygons is a solution, namely it admits a routing of the demand matrix D, then:*

$$\forall i \leq W, \quad D(P_i) \geq i.$$

Proof. Assume a nested sequence of polygons $P_1, P_2, ...$ is a solution. Consider any routing admitted by this solution and any edge $e_i \in P_i$. This edge is contained in exactly $i-1$ edges $e_1 \in P_1, e_2 \in P_2, ..., e_{i-1} \in P_{i-1}$. All the demands routed on the edges $e_1, e_2, ..., e_{i-1}$ could be potentially routed on e_i too. Therefore the sum of the demands that could potentially routed on e_i is at least the sum of the demands actually routed on these edges, which is their total capacity, namely i. Therefore, $D(e_i) \geq i$. This is true for any edge $e_i \in P_i$, we conclude $D(P_i) \geq i$. □

Lemma 4. *If a nested sequence of polygons satisfies:*

$$\forall i \leq W, \quad D(P_i) \geq i.$$

then there is a routing of the uniform demand matrix D.

Proof. We present an algorithm constructing the claimed routing and prove its correctness:

```
RandomRoute(Demand d, Polygon P){
// Routes as much as possible of the demand matrix d
// over the polygon P
    ∀e ∈ P, f(e) = 0
    For each pair of nodes u, v {
        Let e_1, e_2, ..., e_k be the edges of P which are on the shortest
        path from u to v
        and e_i = (a_i, b_i)
        if (k > 0) {
            For (i = 1; i ≤ k; i++) {
                x_i = min(d(u, v), 1 − f(e_i))
                f(e_i) += x_i
                d(a_i, b_i) += d(u, v) − x_i
            }
            d(u, a_1) += d(u, v)
            d(b_k, v) += d(u, v)
            d(u, v) = 0
        }
    }
}
```

```
Route(Demand D){
   d=D;
   for {i = 1; i ≤ W; i++}{
      RandomRoute(d,P_i);
   }
}
```

Claim. If RandomRoute(d,P) is invoked when $d(P) \geq 1$, upon its return $\forall e \in P, f(e) = 1$.

Proof. Consider any edge $e \in P$. $d(P) \geq 1$, therefore $d(e) = \sum_{i,j} d(i,j) \geq 1$. Each pair contributing to this sum is considered exactly once for this edge by the algorithm. The value of $d(i,j)$ when it is considered by the algorithm is at least equal to its value in the beginning of RandomRoute. This is because $d(i,j)$ is decreased only after it is considered. It contributes $d(i,j)$ to $f(e)$ until $f(e) = 1$. The total contribution to $f(e)$ is therefore $\min(1, d(e)) = 1$. $f(e)$ does not decrease through RandomRoute, which means that the value of $f(e)$ upon return is 1.

Claim. If RandomRoute(d,P) is invoked when $d(P) \geq 1$, upon its return $d(P')$ is decremented by 1 for all polygons P' nested in P.

Proof. Consider a polygon P' nested in P and an edge $e' \in P'$. There is exactly one edge $e \in P$ containing e'. All other edges of P are disjoint to e'. A decrease in $d(e')$ occurs only when $d(u,v)$ changes for some pair u, v. This is done always with a change (increase) in $f(e)$ or $f(e'')$ where e'' is a disjoint edge.

Case 1, The change is in $f(e)$: Whenever $f(e)$ is increased, $d(e')$ is decreased by the same amount (see Figure 3). By the previous claim these decreases sum up to 1.

Case 2, The change is in a disjoint edge e'': This change will not affect $d(e')$ because the corresponding increase in the demand $d(b'', v)$ (see Figure 4).

Therefore, $d(e')$ is decreased exactly by 1, and consequently, so is $d(P')$.

By induction on W, using the above results, we prove that: If $\forall i \leq W, D(P_i) \geq i$, the above algorithm ends with $f(e) = 1$ for all the edges. Therefore the nested sequence of polygons is a feasible solution. □

Lemma 3 and Lemma 4 imply:

Theorem 1. *A nested sequence of polygons $P_1, P_2, ..., P_W$ is a solution if and only if*

$$\forall i \leq W, \quad D(P_i) \geq i. \tag{1}$$

5.3 Optimum Solution for W=2

As previously stated, any solution contains the circle $(0, 1, ..., N, 0)$. Furthermore, we know that this solution consists of two convex polygons, namely, the above circle and one convex polygon. The circle is nested in all convex polygons,

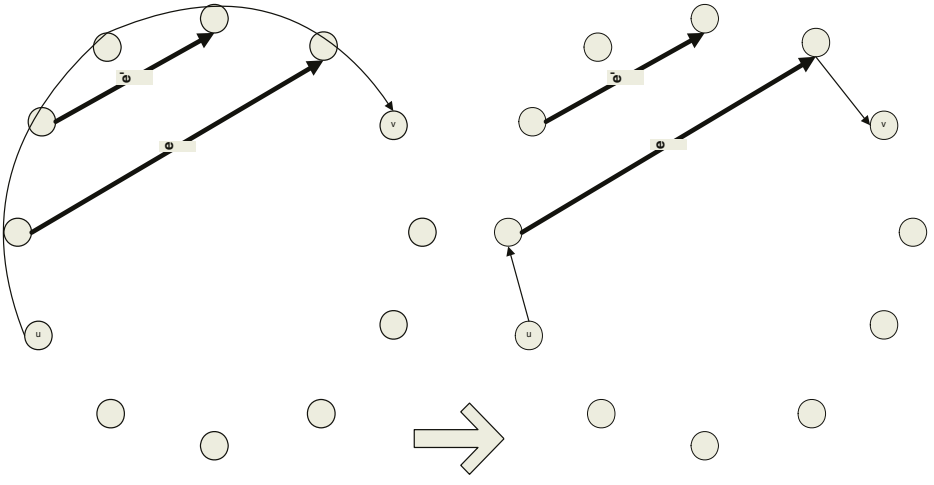


Fig. 3. Change in $f(e)$

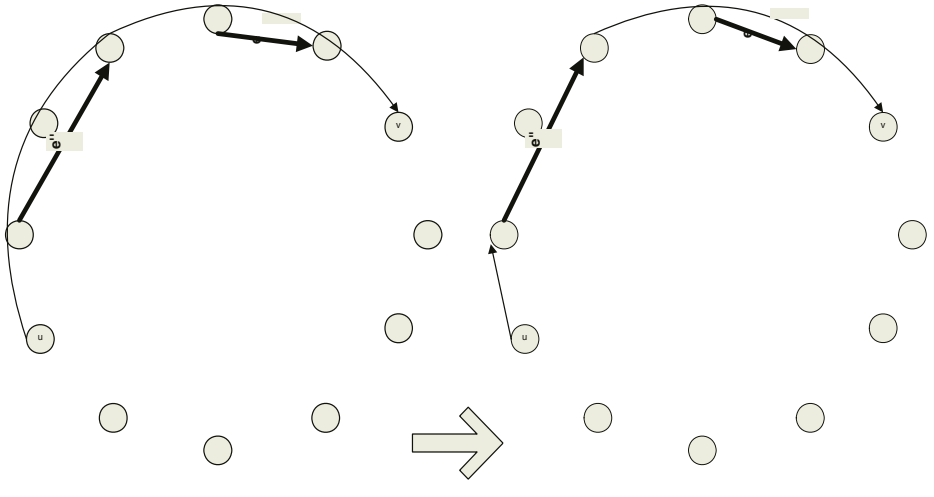


Fig. 4. Change in a disjoint edge

therefore any solution for $W = 2$ is a nested sequence of polygons P_1, P_2 where P_2 is the circle itself. It remains to find P_1:

We know that P_1 must satisfy $D(P_1) \geq 1$. Therefore all the edges $e \in P_1$ must satisfy $D(e) \geq 1$, implying $l(e) \leq D^{-1}(1)$. We want to find the polygon with minimum total number of edges, therefore we choose $l = \lfloor D^{-1}(1) \rfloor$ and

build the polygon which consists of $\lceil 2n/l \rceil$ edges such that $l(e) = l$ and at most one edge such that $l(e) < l$. The number of edges in this polygon is

$$\lceil \frac{2n}{\lfloor D^{-1}(1) \rfloor} \rceil \leq \lceil \frac{2n}{\lfloor \widetilde{D}^{-1}(1) \rfloor} \rceil = \left\lceil \frac{2n}{\left\lfloor (1 - \frac{1}{\sqrt{2}})n \right\rfloor} \right\rceil \approx \frac{2}{1 - \frac{1}{\sqrt{2}}} \approx 6.8$$

This solution uses $N+7$ ADMs instead of $2N$ ADMs in PPWDM, still getting the same throughput. From the above discussion it follows that:

Theorem 2. *For $W = 2$ any optimum solution uses $N + 7$ ADMs.*

5.4 A Solution for any W Using Nested Polygons

Assume for simplicity that N is a power of 2. We build a nested sequence of polygons P_i as follows: All the polygons in the sequence, share a special node 0, and the lengths of their edges are powers of two.

For each $1 \leq i < W$,

$$k_i = \begin{cases} \lfloor \log \widetilde{D}^{-1}(i) \rfloor & \text{if } \widetilde{D}^{-1}(i) \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

All the edges of each polygon P_i have length $l_i = 2^{k_i}$. Clearly, k_i is a non increasing sequence. Therefore the lengths l_i, l_{i+1} of the edges of two consecutive polygons P_i, P_{i+1} satisfy $l_i = m\, l_{i+1}$, where m is some (possibly 0) power of 2. It is clear that this is a nested sequence of polygons. Moreover the lengths satisfy:

$$\begin{aligned} l_i = 2^{k_i} &\leq \widetilde{D}^{-1}(i) \\ \widetilde{D}(l_i) &\geq i \end{aligned}$$

Therefore $D(P_i) \geq i$ which is a sufficient condition for the solution to be feasible (1).

Theorem 3. *The number of ADMs used by the solution is at most $8WlnW + 2N + O(W)$.*

Proof. The number ADM_i of the ADMs of polygon P_i is: $ADM_i = \frac{N}{2^{k_i}}$. For every i such that $\widetilde{D}^{-1}(i) \geq 1$ we have:

$$ADM_i = \frac{2n}{2^{\lfloor \log \widetilde{D}^{-1}(i) \rfloor}} < \frac{2n}{2^{(\log \widetilde{D}^{-1}(i) - 1)}} = \frac{4n}{\widetilde{D}^{-1}(i)} = \frac{4}{\left(1 - \frac{\sqrt{i}}{\sqrt{W}}\right)} = \frac{4\sqrt{W}}{\sqrt{W} - \sqrt{i}}$$

For other values of i we have $ADM_i = N$. Note that, $\widetilde{D}^{-1}(i) \geq 1$ if and only if $i \geq \widetilde{D}(1) = W(1 - 1/n)^2$. The total number of ADMs satisfies:

$$\sum_{i=1}^{W} ADM_i \leq 4\sqrt{W} \sum_{i=1}^{\lfloor W(1-1/n)^2 \rfloor - 1} \frac{1}{\sqrt{W} - \sqrt{i}} + \sum_{i=\lfloor W(1-1/n)^2 \rfloor}^{W} N$$

The first sum above is bounded by

$$
\begin{aligned}
&4\sqrt{W}\sum_{i=1}^{W-2}\frac{1}{\sqrt{W}-\sqrt{i}} \le 4\sqrt{W}\int_{1}^{W-1}\frac{1}{\sqrt{W}-\sqrt{x}}\,dx \\
&= 8\sqrt{W}\left(\sqrt{W}\ln(\sqrt{W}-\sqrt{x})+\sqrt{x}\right)_{W-1}^{1} \\
&= 8W\ln(\sqrt{W}-1) - 8W\ln(\sqrt{W}-\sqrt{W-1}) + 8\sqrt{W}(1-\sqrt{W-1}) \\
&\le 8W\ln\frac{\sqrt{W}-1}{\sqrt{W}-\sqrt{W-1}} \\
&= 8W\ln(\sqrt{W}-1)(\sqrt{W}+\sqrt{W-1}) \\
&< 8W\ln(2W) \\
&= 8W\ln W + O(W)
\end{aligned}
$$

and the second sum is bounded by:

$$
\begin{aligned}
&N(W - W(1-\frac{1}{n})^2 + 2) \\
&= N(W(1-(1-\frac{1}{n})^2)+2) = N(W\frac{1}{n}(2-\frac{1}{n})+2) \\
&< 4W + 2N
\end{aligned}
$$

Summing both bounds we get:

$$\sum_{i=1}^{W} ADM_i \le 8WlnW + 2N + O(W)$$

□

Conclusion: If $W = O(N/\log N)$ the asymptotic cost depends only on N, and if $W = \Omega(N^{1+\epsilon}/\log N)$ for any $\epsilon > 0$, then the asymptotic cost depends only on W.

5.5 An Improved Upper Bound

Now we show how the $O(W \log W + N)$ upper bound of Theorem 3 can be further improved.

Lemma 5. *The problem is sub-additive in W.*

Proof. A solution S_1 for W_1 wavelengths and N nodes and a solution S_2 for W_2 wavelengths and N nodes can be superposed to obtain a solution for $W_1 + W_2$ wavelengths. This is true because:

$$\tau = \frac{2(W_1+W_2)}{n^2} = \frac{2W_1}{n^2} + \frac{2W_2}{n^2} = \tau_1 + \tau_2$$

where τ (resp. τ_1, τ_2) are the uniform demands for W (resp. W_1, W_2) wavelengths. □

Theorem 4. *There is a solution using* $O(W \log \overline{W} + N)$ *ADMs, where* $\overline{W} = o(W)$.

Proof. We omit constant factors. We consider two cases:

- $W \log W$ is $O(N)$. In this case our upper bound is $O(N)$ which is optimal.
- N is $o(W \log W)$. In this case let $\overline{W}$ be such that $N = \overline{W} \log \overline{W}$. $\overline{W}$ is $o(W)$. Let $X = W/\overline{W}$. Because of the sub-additivity, the superposition of X solutions of an instance with $\overline{W}$ wavelengths and N nodes is a solution for our instance of W wavelengths and N nodes. This solution uses

$$X \cdot O(\overline{W} \log \overline{W} + N) = X \cdot O(N) = O(\frac{W}{\overline{W}} \overline{W} \log \overline{W}) = O(W \log \overline{W})$$

ADMs.

□

6 Conclusion and Open Problems

Our result is a first attempt to understand the trade-off between the wavelength minimization and ADM minimization problem. We dealt with the special case of splittable, multi-hop communication. We considered the all-to-all uniform traffic instance. In cases that the requests are not splittable, our solution can be used for infinitely many values of W and N, namely for the cases where $2W/n^2$ is an integer.

An $O(W + N)$ lower bound is immediate. The problem of closing the gap between our $O(W \log \overline{W} + N)$ solution and this lower bound remains open. Another open question is whether the problem considered is NP-hard.

Acknowledgement. We thank Prof. Reuven Cohen and Doron Tsur for introducing us to the problem and for helpful discussions.

References

1. O. Gerstel, P. Lin, and G. Sasaki. Combined wdm and sonet network design. In *INFOCOM'99, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 734–43, 1999.
2. O. Gerstel, P. Lin, and G. Sasaki. Wavelength assignment in a wdm ring to minimize cost of embedded sonet rings. In *Infocom'98, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 69–77, 1998.
3. L. Liu, X. Li, P-J. Wan, and O. Frieder. Wavelength assignment in a wdm rings to minimize sonet adms. In *INFOCOM'2000, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Tel-Aviv, Israel*, pages 1020–1025, 2000.

4. G. Wilfong and P. Winkler. Ring routing and wavelength translation. In *Proceedings of the ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'98, San Francisco, California*, pages 333–341, Jan 1998.
5. S. Khanna. A polynomial time approximation scheme for the sonet ring loading problem. *Bell Labs Technical Journal*, pages 36–41, Spring 1997.
6. T. Eilam, S. Moran, and S. Zaks. Lightpath arrangement in survivable rings to minimize the switching cost. *IEEE Journal of Selected Area on Communications*, 20(1):172–182, Jan 2002.
7. G. Călinescu and P-J. Wan. Traffic partition in wdm/sonet rings to minimize sonet adms. In *15th International Parallel and Distributed Processing Symposium*, 2001.
8. M. Shalom and S. Zaks. 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.
9. L. Epstein and A. Levin. Better bounds for minimizing sonet adms. In *WAOA, Workshop on Approximation and Online Algorithms*, Sep 2004.
10. P.J. Wan, G. Călinescu, L.-W. Liu, and O. Frieder. Grooming of arbitrary traffic in sonet/wdm rings. *IEEE Journal of Selected Area on Communications*, 18(10):1995–2003, 2000.

Optimal Gossiping in Square Meshes in All-Port Mode and with Short Packets

(Extended Abstract)

Rui Wang* and Francis C.M. Lau

Department of Computer Science, The University of Hong Kong

Abstract. We present optimal solutions for the gossiping problem in square meshes under the all-port, full-duplex (F*) and the all-port, half-duplex (H*) models. We assume packets are of finite size, each carrying at most one message.

Keywords: Gossiping, meshes, information dissemination, networks, algorithms.

1 Introduction

Gossiping, also known as total exchange and all-to-all (nonpersonalized) broadcast, is a communication problem in which each processor (or node) has a unique message to be transmitted to every other processor. Because of its rich communication pattern, gossiping is a useful benchmark for evaluating the communication capability of an interconnection structure. The gossiping problem has been studied extensively during the last two decades; a summary of the results can be found in [7], [9], [10]. Gossiping is needed in many communication scenarios, including those arising from the use of radio networks [4].

Krumme et al. have suggested that the gossiping problem can be studied under four different communication models [11]: (1) the full-duplex, all-port model, (2) the full-duplex, one-port model, (3) the half-duplex, all-port model, and (4) the half-duplex, one-port model, which can be identified by the labels F*, F1, H*, H1, respectively. A full-duplex link allows both ends to send/receive a message at the same time; a half-duplex link allows only one end to do so at a time. In the one-port mode, only one of the incident links of a node may be active at a time; all the incident links may be active at the same time in the all-port mode. The four models, therefore, form a spectrum, with F* being the strongest in communication capability and H1 the weakest.

Bermond et al. [3] have added another dimension to the problem, suggesting that a packet carrying messages cannot be of infinite size. They studied the

* Correspondence: Rui Wang, Department of Computer Science, The University of Hong Kong, Hong Kong / Email: rwang@cs.hku.hk.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 292–306, 2005.

gossiping problem under this hypothesis and under the F1 model, deriving results for the complete graph, hypercube, cycle, and path [5]. Bagchi et al. have considered the same, but under the H1 model [1], [2].

We study gossiping schemes with the bounded packet size restriction. We use p to denote the size of a packet: $p = 1$ means that a packet can carry up to one message, $p = 2$ two messages, etc.

The gossiping process advances by rounds (or timestep) in a lock-step fashion. In each round, a packet can only travel across one edge. We define $g_{F*}(N)$ and $g_{H*}(N)$ to be the times (rounds) of the F* and H* models required respectively to complete a gossip for the interconnection network N with $p = 1$. Our focus is on the 2D mesh which is an important communication fabric for modern parallel machines including some of the fastest machines in the Top500 list. Under the restriction of bounded packets size, the following are the existing results.

F* model: For $p = 1$, Soch and Tvrdik obtained the optimal result, $\lceil (mn - 1)/2 \rceil$, for the $m \times n$ mesh with the restriction that at least one of m and n must be even [15].

F1 model: For $p = 1$, Bermond et al. gave an algorithm that can solve the problem for the $m \times n$ mesh in $2mn - 3 - max\{m, n\}$ steps, m and n odd [3].

H* model: For $p = 1$, Fujita and Yamashita gave an algorithm that can solve the problem for $n \times n$ (square) mesh in $n(n+1)/2 + \lfloor (3n-5)/2 \rfloor$ steps [8]. Based on the optimal gossiping in path given in [13], Lau and Zhang [12] have improved this to $n(n+1)/2 + \lfloor (n-1)/2 \rfloor$.

H1 model: For any value of p, Bagchi et al. derived the result $2mn/p + O(m + n)$, for the $m \times n$ mesh [2].

In this paper, we assume the all-port mode and consider only the case of $p = 1$. Many of the modern designs of routers use separate controllers to manage the links, which can operate simultaneously in parallel. Both the F* and the H* model are realistic models for router implementation. One well-known example of H* router is the Network Design Frame [6]. The C104 router for transputer [14] is an example of the F* model.

We present optimal algorithms for $n \times n$ square meshes (n can be odd or even) under the H* and the F* models with $p = 1$. The algorithms gossip along shortest paths.

2 Preliminaries

An $n \times n$ square mesh $M_{n \times n} = (V, E)$ is a graph of $|V| = n^2$ vertices and $|E| = 2n(n-1)$ edges, and has n rows and n columns. Existing lower bounds for square meshes as given in [8], [12], and [13] are as follows.

Lemma 2.1. *$g_{H*}(M_{n \times n}) \geqslant n(n+1)/2$, and $g_{F*}(M_{n \times n}) \geqslant \lceil (n^2 - 1)/2 \rceil$.*

This paper shows that both bounds are tight.

We number the rows (the columns) of $M_{n\times n}$ with $-k, -k+1, \ldots, -1, 0, 1, \ldots k-1, k$ from top to bottom (from left to right) if $n = 2k+1$ is odd. If n is even $(2k)$, we omit row and column 0. A node (vertex) is uniquely indexed by $v_{x,y}$, where x and y are the node's row and column numbers, respectively. A mesh $M_{n\times n} = (V, E)$, in which each vertex $v_{x,y} \in V$ initially (at round 0) holds a unique message $\sigma_{x,y}$, begins gossiping at round 1. At the end of gossiping, every vertex has all the messages. We use $(-1, 0)$, $(1, 0)$, $(0, -1)$ and $(0, 1)$ to refer to the up, down, left, and right directions, respectively, for message movements. The set of the messages sent to $v_{x,y}$ via the edge $(v_{x-a,y-b}, v_{x,y})$, i.e., along direction (a, b), is denoted by $P^{a,b}_{x,y}$. Note that the zero direction $(0, 0)$ is also involved and $P^{0,0}_{x,y} = \{\sigma_{x,y}\}$. We assume that a message will not reach the same node more than once so that we can define $R_{x,y}(\sigma_{i,j})$ to be the round at which the message $\sigma_{i,j}$ arrives at $v_{x,y}$. A gossiping algorithm can be completely represented by the pair (P, R).

Definition 2.1. *An F* Gossiping Scheme* (GS) *on mesh* $M_{n\times n} = (V, E)$ *is a pair* (P, R)*, such that for* $k \leqslant x, y \leqslant k$ *and directions* $(a, b), (c, d)$*,*

***complete*:** $P^{0,0}_{x,y} \cup P^{0,1}_{x,y} \cup P^{1,0}_{x,y} \cup P^{0,-1}_{x,y} \cup P^{-1,0}_{x,y} = \{\sigma_{i,j} | v_{i,j} \in V\}$*, i.e., every message will eventually reach* $v_{x,y}$*;*
***initialized*:** $P^{0,0}_{x,y} = \{\sigma_{x,y}\}$ *and* $R_{x,y}(\sigma_{x,y}) = 0$*;*
***1-bounded*:** $\sigma, \sigma' \in P^{a,b}_{x,y}, \sigma \neq \sigma' \Longrightarrow R_{x,y}(\sigma) \neq R_{x,y}(\sigma')$*, i.e.,* $p = 1$*;*
***duplication free*:** $(a, b) \neq (c, d) \Longrightarrow P^{a,b}_{x,y} \cap P^{c,d}_{x,y} = \phi$*;*
***precedence constrained*:** $(a, b) \neq (0, 0), \sigma \in P^{a,b}_{x,y} \Longrightarrow R_{x,y}(\sigma) > R_{x-a,y-b}(\sigma)$*, i.e., a message leaving a node must have first arrived at the node.*

The H* model gossiping scheme needs a further restriction:

Definition 2.2. *An F* GS* (P, R) *is also an H* GS if all links are* ***half-duplex****:* $\sigma \in P^{a,b}_{x,y}, \sigma' \in P^{-a,-b}_{x-a,y-b} \Longrightarrow R_{x,y}(\sigma) \neq R_{x-a,y-b}(\sigma')$.

The quality of a GS (P, R) is measured by $max\{R_{x,y}(\sigma_{i,j}) | v_{x,y}, v_{i,j} \in V\}$, the time it requires to complete the gossip. If this number is no more than r, the gossiping scheme is termed an r-GS. Clearly, on a mesh of size $n \times n$, $r \geqslant \lceil (n^2 - 1)/2 \rceil$, and $r \geqslant n(n+1)/2$ if the GS is of the H* model.

Rotating a direction (a, b) by 90° clockwise gives the direction $(b, -a)$; rotating a square mesh by 90° clockwise will superimpose node $v_{x,y}$ upon $v_{y,-x}$. Square meshes are highly symmetric with respect to rotating. An ideal GS should be that everything looks the same from the different sides. This can be captured by a definition.

Definition 2.3. *A statement* $Q((x_1, y_1), (x_2, y_2), \ldots, (x_z, y_z))$*, where each* (x_i, y_i) *is either a node index or a direction, is* 90°*-rotating symmetric if* $Q((x_1, y_1), (x_2, y_2), \ldots, (x_z, y_z)) \Longleftrightarrow Q((y_1, -x_1), (y_2, -x_2), \ldots, (y_z, -x_z))$.

The rotating symmetry says that $\sigma_{i,j} \in P^{a,b}_{x,y} \Longleftrightarrow \sigma_{j,-i} \in P^{b,-a}_{y,-x}$, and $R_{x,y}(\sigma_{i,j}) = r \Longleftrightarrow R_{y,-x}(\sigma_{j,-i}) = r$. That is, if the mesh is rotated clockwise by 90°, $P^{1,0}_{x,y}$ will completely overlap with $P^{0,-1}_{y,-x}$, and so on.

With rotating symmetry we need only to consider $P^{a,b}_{x,y}$ and $R_{x,y}(\sigma_{i,j})$ for $-k \leqslant x, y \leqslant 0$. We cannot expect more symmetry than rotating symmetry, such as "flipping"; for example, if $\sigma_{i,j} \in P^{a,b}_{x,y} \iff \sigma_{j,i} \in P^{b,a}_{y,x}$ (flipping along the diagonal) also holds, then message $\sigma_{i,i}$ will be in two different $P^{a,b}_{0,0}$'s, violating the *duplication free* requirement.

We present an F* $\lceil (n^2-1)/2 \rceil$-GS and an H* $(n(n+1)/2)$-GS for $M_{n\times n}$—both are optimal, rotating symmetric, and shortest-path routed. Section 3 focuses on odd n, and Section 4 on even n. What is interesting is that the H* and F* optimal schemes can share the same P and the same order for messages arriving at the nodes. Same P means identical routing paths, and same arriving order means that the i-th messages passing through a same link under the different models are the same ones.

3 Odd n

Throughout this section, we assume $n = 2k+1$ with $k \geqslant 1$ and consider $M_{n\times n}$. Section 3.1 focuses on the F* model, presenting an optimal F* GS (P, R). We will construct a new P in Section 3.2 and an R^h in 3.3 such that (P, R^h) is an optimal H* GS. Denoting by $R^f_{x,y}(\sigma)$ the arriving order of σ at $v_{x,y}$ under GS (P, R^h), (P, R^f) forms an optimal F* GS.

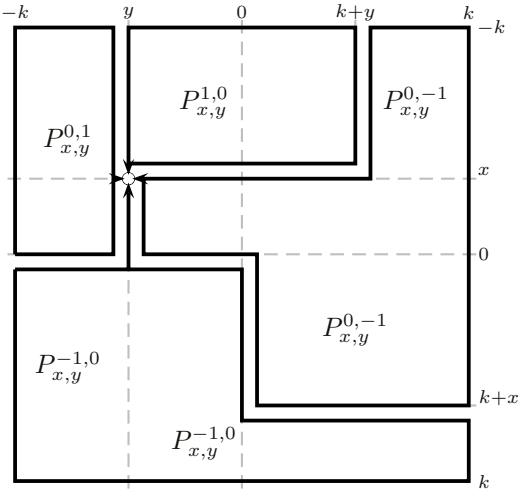


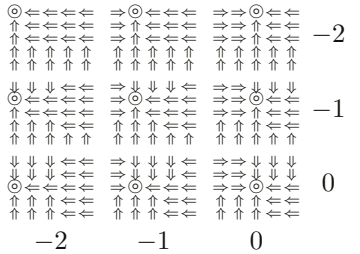
Fig. 1. Message gathering at $v_{x,y}$ for $-k \leqslant x, y \leqslant 0$. Rotating it and changing the labels accordingly gives the scenarios of the other areas

3.1 An F* GS

We choose P such that $|P^{a,b}_{x,y}| = (k + ax + by)(k + 1)$. By the 90°-rotating symmetry, we need only to determine $P^{a,b}_{x,y}$ for $-k \leqslant x, y \leqslant 0$. Our design is illustrated in Fig. 1, where

$$\begin{aligned}
P^{0,0}_{x,y} &= \{\sigma_{x,y}\},\\
P^{0,1}_{x,y} &= \{\sigma_{i,j} | -k\leqslant i\leqslant 0, -k\leqslant j<y\},\\
P^{1,0}_{x,y} &= \{\sigma_{i,j} | -k\leqslant i<x, y\leqslant j\leqslant k+y\},\\
P^{0,-1}_{x,y} &= \{\sigma_{i,j} | x\leqslant i\leqslant k+x, 0<j\leqslant k\} \cup \{\sigma_{i,j} | x<i\leqslant 0, y<j\leqslant 0\} \cup \{\sigma_{i,j} | -k\leqslant i<x, k+x\leqslant j\leqslant k\},\\
P^{-1,0}_{x,y} &= \{\sigma_{i,j} | 0<i\leqslant k, -k\leqslant j\leqslant 0\} \cup \{\sigma_{i,j} | x<i\leqslant k, j=y\} \cup \{\sigma_{i,j} | k+x<i\leqslant k, 0<j\leqslant k\}.
\end{aligned} \tag{3.1}$$

Fig. 2 gives an example of a complete GS, for $M_{5\times 5}$. Clearly, the above P is duplication free. Note that $|P^{0,0}_{x,y}| = 1$ and



◎ *01 02 03 04* *01* ◎ *01 02 03* *02 01* ◎ *01 02*
01 *11 09 07 05* *02* **01** *08 06 04* *03 05* **01** *05 03*
02 *12 10 08 06* *03* **02** *09 07 05* *04 06* **02** *06 04*
03 07 08 12 11 **07 03 08 12 11** **08 07 03 12 11**
04 05 06 10 09 **05 04 06 10 09** **06 05 04 10 09**

01 02 03 *12 10* *02* 01 02 03 *09* *03 05* 01 02 03
◎ *01 02 03 04* *01* ◎ *01 02 03* *02 01* ◎ *01 02*
01 *11 09 07 05* *03* **01** *08 06 04* *04 06* **01** *05 03*
02 06 07 *08 06* **06 02 07** *07 05* **07 06 02** *06 04*
03 04 05 09 08 **04 03 05 09 08** **05 04 03 09 08**

02 03 04 *11 09* *03* 02 03 04 *08* *04 06* 02 03 04
01 05 06 *12 10* *02* 01 05 06 *09* *03 05* 01 05 06
◎ *01 02 03 04* *01* ◎ *01 02 03* *02 01* ◎ *01 02*
01 05 06 *07 05* **05 01 06** *06 04* **06 05 01** *05 03*
02 03 04 *08 06* **03 02 04** *07 05* **04 03 02** *06 04*

Fig. 2. $P^{a,b}_{x,y}$ and $R_{x,y}(\sigma_{i,j})$ in $M_{5\times5}$ for $-2 \leqslant x, y \leqslant 0$. The l.h.s. is for $P^{a,b}_{x,y}$, in which there are $3 \times 3 = 9$ matrices, each for one node $v_{x,y}$ in the region $-2 \leqslant x, y \leqslant 0$; the one for $v_{x,y}$ shows the gathering of messages at $v_{x,y}$. In each matrix, message $\sigma_{i,j}$ is represented by an arrow in position (i, j) to indicate the membership of $\sigma_{i,j}$ to $P^{a,b}_{x,y}$. The arrows $\Downarrow$, $\Leftarrow$, $\Uparrow$ and $\Rightarrow$ are for sets $P^{-1,0}_{x,y}$, $P^{0,-1}_{x,y}$, $P^{1,0}_{x,y}$ and $P^{0,1}_{x,y}$ respectively. On the r.h.s., the numbers indicate both the rounds and the $P^{a,b}_{x,y}$ they belong, different fonts for different $P^{a,b}_{x,y}$'s. To see the scenario at nodes other than $-2 \leqslant x, y \leqslant 0$, simply rotate the figure

$$|P^{a,b}_{x,y}| = (k + ax + by)(k + 1) = |P^{a,b}_{x-a,y-b}| + k + 1 \quad if\ (a, b) \neq (0, 0) \quad (3.2)$$

which implies that $|P^{0,0}_{x,y}| + |P^{0,1}_{x,y}| + |P^{1,0}_{x,y}| + |P^{0,-1}_{x,y}| + |P^{-1,0}_{x,y}| = n^2$. Combining this with duplication freeness, completeness would follow, and thus P is qualified to be a part of the gossiping scheme. Since $-k \leqslant ax + by \leqslant k$, we have

$$|P^{a,b}_{x,y}| \leqslant 2k(k + 1) = \frac{n^2 - 1}{2}.$$

Hence, P is possible to support an F* GS to match the lower bound $(n^2 - 1)/2$. To achieve this, we are going to define an R that lets the messages of $P^{a,b}_{x,y}$ arrive at $v_{x,y}$ respectively in rounds $1, 2, \ldots, |P^{a,b}_{x,y}|$, i.e.,

$$\left\{R_{x,y}(\sigma_{i,j}) | \sigma_{i,j} \in P^{a,b}_{x,y}\right\} = \begin{cases} \{0\} & if\ (a, b) = (0, 0), \\ \{1, 2, \ldots, |P^{a,b}_{x,y}|\} & if\ (a, b) \neq (0, 0). \end{cases} \quad (3.3)$$

Obviously, if we did so, we need not worry about the 1-bounded requirement and the optimality, but only to guarantee the precedence constraint. For this, we rewrite (3.1) in a recurrence form.

$$\begin{aligned} P^{0,0}_{x,y} &= \{\sigma_{x,y}\}, \\ P^{0,1}_{x,y} &= P^{0,0}_{x,y-1} \cup P^{0,1}_{x,y-1} \cup \{\sigma_{i,y-1} | -k \leqslant i < x\} \cup \{\sigma_{i,y-1} | x < i \leqslant 0\}, \\ P^{1,0}_{x,y} &= P^{0,0}_{x-1,y} \cup P^{1,0}_{x-1,y} \cup \{\sigma_{x-1,j} | y < j \leqslant k + y\}, \\ P^{0,-1}_{x,y} &= P^{0,0}_{x,y+1} \cup P^{0,-1}_{x,y+1} \cup \{\sigma_{i,y+1} | x < i \leqslant 0\} \cup \{\sigma_{i,k+y+1} | -k \leqslant i < x\}, \\ P^{-1,0}_{x,y} &= P^{0,0}_{x+1,y} \cup P^{-1,0}_{x+1,y} \cup \{\sigma_{k+x+1,j} | 0 < j \leqslant k\}. \end{aligned} \quad (3.4)$$

For $(a, b) \neq (0, 0)$, the above can be summarized as

$$P^{a,b}_{x,y} = P^{0,0}_{x-a,y-b} \cup P^{a,b}_{x-a,y-b} \cup Q^{a,b}_{x,y}, \tag{3.5}$$

where $Q^{a,b}_{x,y}$ is a set of k messages from $P^{b,-a}_{x-a,y-b}$ and/or $P^{-b,a}_{x-a,y-b}$. With this observation, we let $v_{x,y}$ receive $P^{a,b}_{x,y}$ from $v_{x-a,y-b}$ in the order: $P^{0,0}_{x-a,y-b}$ first, then $P^{a,b}_{x-a,y-b}$, and $Q^{a,b}_{x-a,y-b}$ the last. Within each set, we adopt "the first arriving at $v_{x-a,y-b}$, the first to be received by $v_{x,y}$, breaking ties arbitrarily". To be precise, besides $R_{x,y}(\sigma_{x,y}) = 0$ as an initialization, we have

$$R_{x,y}(\sigma_{i,j}) = R_{x-a,y-b}(\sigma_{i,j}) + 1 \quad \textit{for } \sigma_{i,j} \in P^{0,0}_{x-a,y-b} \cup P^{a,b}_{x-a,y-b}, \tag{3.6}$$

$$\{R_{x,y}(\sigma_{i,j})|_{\sigma_{i,j} \in Q^{a,b}_{x,y}}\} = \left\{ |P^{a,b}_{x-a,y-b}| + 1 + r \middle| 1 \leqslant r \leqslant k \right\}. \tag{3.7}$$

Thus, $v_{x,y}$ receives from $v_{x-a,y-b}$ the loner $\sigma_{x-a,y-b}$ of $P^{0,0}_{x-a,y-b}$ in the first round, and then in rounds 2 to $|P^{a,b}_{x-a,y-b}| + 1$, collects $P^{a,b}_{x-a,y-b}$, and at last the k messages of $Q^{a,b}_{x,y}$ in rounds $|P^{a,b}_{x-a,y-b}| + 2$ to $|P^{a,b}_{x-a,y-b}| + k + 1 = |P^{a,b}_{x,y}|$, satisfying (3.3). This can be seen in the example in Fig. 2.

About the precedence constraint, for $\sigma_{i,j} \in P^{0,0}_{x-a,y-b} \cup P^{a,b}_{x-a,y-b}$, (3.6) guarantees that R conforms to $R_{x,y}(\sigma_{i,j}) > R_{x-a,y-b}(\sigma_{i,j})$. So, we need only to justify that the messages in $Q^{a,b}_{x,y}$ can be received in the rounds specified in (3.7) without breaking the precedence constraint. We need a simple fact from (3.5) and (3.6); it is for $(a,b) \neq (0,0)$ and $r \geqslant 0$:

$$\sigma_{i,j} \in P^{0,0}_{x-ar,y-br} \cup P^{a,b}_{x-ar,y-br} \Longrightarrow R_{x,y}(\sigma_{i,j}) = R_{x-ar,y-br}(\sigma_{i,j}) + r. \tag{3.8}$$

Applying this to $\sigma_{x-ra,y-rb} \in P^{0,0}_{x-ar,y-br}$ gives that $R_{x,y}(\sigma_{x-ar,y-br}) = r$, or $R_{x,y}(\sigma_{x\pm r,y}) = R_{x,y}(\sigma_{x,y\pm r}) = r$ for $r \geqslant 0$.

By the rotating symmetry, the discussion can be restricted to the following four cases:

$\boldsymbol{Q^{0,1}_{x,y},\ -k \leqslant x, y \leqslant 0}$**:** By (3.8), messages of $\{\sigma_{i,y-1}|_{-k \leqslant i < x}\}$ reach $v_{x,y-1}$ in rounds $1, 2, \ldots k - |x|$, and those of $\{\sigma_{i,y-1}|_{x < i \leqslant 0}\}$ in rounds $1, 2, \ldots, |x|$. So, at $v_{x,y}$, the former can be assigned rounds from $|P^{0,1}_{x,y-1}| + 1 + 1$ to $|P^{0,1}_{x,y-1}| + 1 + k - |x|$ and the later from $|P^{0,1}_{x,y-1}| + 2 + k - |x|$ to $|P^{0,1}_{x,y-1}| + 1 + k$.

$\boldsymbol{Q^{1,0}_{x,y},\ -k \leqslant x, y \leqslant 0}$**:** For $\{\sigma_{x-1,j}|_{y < j \leqslant k+y}\}$, the argument is similar.

$\boldsymbol{Q^{0,-1}_{x,y}, -k \leqslant x \leqslant 0, -k \leqslant y < 0}$**:** $\{\sigma_{i,y+1}|_{x < i \leqslant 0}\}$ reach $v_{x,y+1}$ in rounds 1 to $|x|$. As $\{\sigma_{i,k+y+1}|_{-k \leqslant i < x}\} \subseteq P^{1,0}_{x,y+1}$, they reach $v_{x,y+1}$ no later than $|P^{1,0}_{x,y+1}| = (k+x)(k+1) \leqslant k(k+1)$. Since $|P^{0,-1}_{x,y}| = (k-y)(k+1) \geqslant (k+1)(k+1) = k(k+1) + k$, the last k rounds of $P^{0,-1}_{x,y}$ can be assigned to $Q^{0,-1}_{x,y}$ without breaking the precedence constraint.

$\boldsymbol{Q^{-1,0}_{x,y}, -k \leqslant x < 0, -k \leqslant y \leqslant 0}$**:** $\{\sigma_{k+x+1,j}|_{0 < j \leqslant k}\} \subseteq P^{0,-1}_{x+1,0}$. By (3.8), for $\sigma \in Q^{-1,0}_{x,y}$, $R_{x+1,y}(\sigma) = R_{x+1,0}(\sigma) + |y| \leqslant |P^{0,-1}_{x+1,0}| + |y| \leqslant k(k+1) + k$. So, the k messages of $Q^{-1,0}_{x,y}$ come to $v_{x+1,y}$ at rounds no later than $k(k+1) + 1, k(k+1) + 2, \ldots, k(k+1) + k$, respectively, and therefore can be forwarded to $v_{x,y}$ at rounds $k(k+1) + 2, k(k+1) + 3, \ldots, k(k+1) + k + 1$, respectively, where $k(k+1) + k + 1 = (k+1)(k+1) \leqslant (k-x)(k+1) = |P^{-1,0}_{x,y}|$.

In any case, all of the k messages in $Q^{a,b}_{x,y}$ can be sent to $v_{x,y}$ from $v_{x-a,y-b}$ in the last k rounds, from $|P^{a,b}_{x,y}| - k$ to $|P^{a,b}_{x,y}|$, in accordance with the precedence constraint.

Finally, P is designed to be such that $\sigma_{i,j} \in P^{a,b}_{x,y}$ with $(a,b) \neq (0,0)$ only if $ai+bj < ax+by$, or $v_{x,y}$ receives from above only those messages that are above it, from below only those below it, from left only those on its left, and from right only those on its right. This promises us a shortest-path gossiping. Now we can conclude:

Theorem 3.1. *For $M_{n\times n}$ where n is odd, (P,R) is an optimal F* GS which completes the gossiping within $(n^2-1)/2$ rounds and routes messages along shortest paths.*

As a closing remark, to extend the GS in general 2D mesh $M_{m\times n}$, where $m = 2k+1, n = 2l+1$ with $k,l \geqslant 1$, a natural generalization is to extend P such that $|P^{a,b}_{x,y}| = (|ak+bl|+ax+by)(|bk+al|+1) \leqslant (mn+|m-n|-1)/2$. Based on this P an $\big((mn+|m-n|-1)/2\big)$-GS in $M_{m\times n}$ can be generated (the messages of $\{\sigma_{i,k+y} | i < x\} \subseteq P^{1,0}_{x,y}$ need to be speeded up to $v_{x,y}$). Comparing with the lower bound $(mn-1)/2$, it uses $|m-n|/2$ rounds more. In fact, the optimality can be reached. There is a P that satisfies $|P^{a,b}_{x,y}| \leqslant (mn-1)/2$ and supports an F* $\big((mn-1)/2\big)$-GS in $M_{m\times n}$. Due to the lack of space, we claim without proof:

Theorem 3.2. *For odd $m,n > 1$, $g_{F*}(M_{m\times n}) = (mn-1)/2$.*

3.2 *P* for Both Models

We now turn to an optimal GS for H* model. To match the lower bound, we must keep every edge busy, i.e., $|P^{a,b}_{x,y}| + |P^{-a,-b}_{x-a,y-b}| = n(n+1)/2$. We adopt (3.2) as the size of $P^{a,b}_{x,y}$. The idea of (3.1), letting $v_{x,y}$ forward all it has from $v_{x-a,y-b}$ to $v_{x+a,y+b}$, however, is not always applicable, because $P^{a,b}_{x,y}$ may have a message arriving at the round $n(n+1)/2$ (either $P^{a,b}_{x,y}$ or $P^{-a,-b}_{x-a,y-b}$ must have such a message), leaving no time for $v_{x,y}$ to forward it. In such a case $v_{x,y}$ has to give up the "dead" message and resort to those coming from other directions to make up. The refined P for the H* (as well as F*) model is shown in Fig. 3,

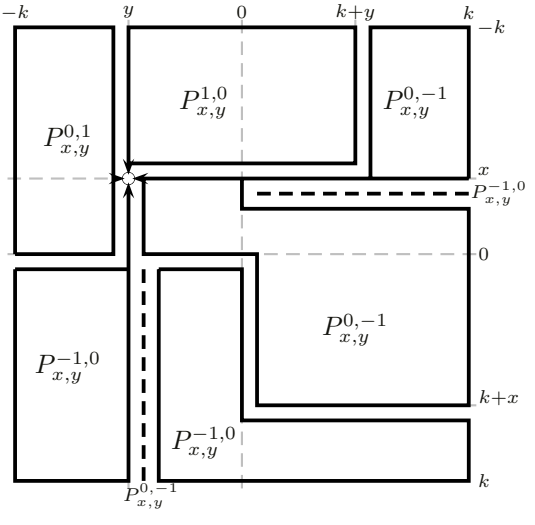


Fig. 3. $P^{a,b}_{x,y}$ for $-k \leqslant x \leqslant 0$ and $-k \leqslant y < 0$

in which the two dashed lines, each covering k messages, belong to $P^{0,-1}_{x,y}$ and $P^{-1,0}_{x,y}$, respectively. Mathematically, we have

$$P^{0,0}_{x,y} = \{\sigma_{x,y}\}; \tag{3.9}$$

$$P^{0,1}_{x,y} = \{\sigma_{i,j}|_{-k\leqslant i\leqslant 0,\ -k\leqslant j<y}\} = P^{0,1}_{x,y-1} \cup \{\sigma_{i,y-1}|_{-k\leqslant i\leqslant 0}\}; \tag{3.10}$$

$$P^{1,0}_{x,y} = \{\sigma_{i,j}|_{-k\leqslant i<x,\ y\leqslant j\leqslant k+y}\} = P^{1,0}_{x-1,y} \cup \{\sigma_{x-1,j}|_{y\leqslant j\leqslant k+y}\}; \tag{3.11}$$

$$\begin{aligned} P^{0,-1}_{x,y} &= P^{0,0}_{x,y+1} \cup P^{0,-1}_{x,y+1} \cup \{\sigma_{i,k+y+1}|_{-k\leqslant i<x}\} \\ &\quad \cup \{\sigma_{i,y+1}|_{x<i\leqslant k}\} - \begin{cases} \{\sigma_{x+1,j}|_{0<j\leqslant k}\} & y=-1, \\ \{\sigma_{i,y+2}|_{0<i\leqslant k}\} & y<-1; \end{cases} \end{aligned} \tag{3.12}$$

$$\begin{aligned} P^{-1,0}_{x,y} &= P^{0,0}_{x+1,y} \cup P^{-1,0}_{x+1,y} \cup \{\sigma_{x+1,j}|_{0<j\leqslant k}\} \\ &\quad \cup \begin{cases} \{\sigma_{x+1,j}|_{-k\leqslant j<y}\} \cup \{\sigma_{x+1,j}|_{y+1<j\leqslant 0}\} - \{\sigma_{i,y+1}|_{1<i\leqslant k}\} & x=0, \\ \{\sigma_{k+x+1,j}|_{0<j\leqslant k}\} - \{\sigma_{x+2,j}|_{0<j\leqslant k}\} & x<0. \end{cases} \end{aligned} \tag{3.13}$$

All the above are for $-k \leqslant x, y \leqslant 0$ and $y \neq 0$, and (3.9) and (3.11) are also for $y = 0$. For nonzero direction (a, b) other than $(1, 0)$, $P^{a,b}_{0,0}$ is implied by (3.11) with the rotating symmetry. Fig. 4 shows an example in $M_{7\times 7}$. It is easy to see that P is complete and duplication free and conforms to (3.2). For $(a, b) \neq (0, 0)$, some facts about $P_{x,y}$ are:

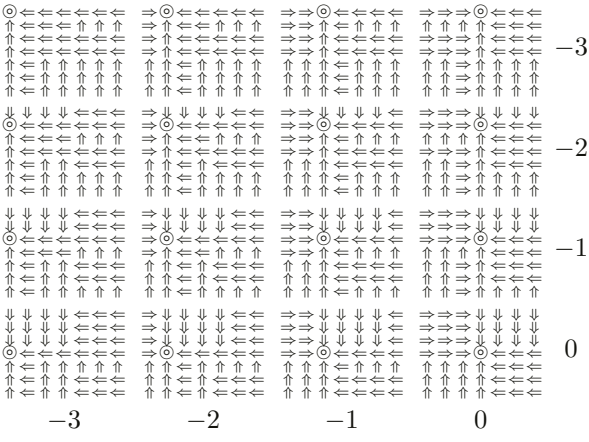


Fig. 4. $P^{a,b}_{x,y}$ in $M_{7\times 7}$ for $-3 \leqslant x, y \leqslant 0$

$$|P^{a,b}_{x,y}| = |P^{a,b}_{x-a,y-b}| + k + 1 = (k + ax + by)(k + 1) \leqslant (n^2 - 1)/2; \tag{3.14}$$

$$|P^{a,b}_{x,y}| + |P^{-a,-b}_{x-a,y-b}| = n(n + 1)/2; \tag{3.15}$$

$$|P^{a,b}_{x,y}| \neq |P^{-a,-b}_{x-a,y-b}| \quad \text{and} \quad |P^{a,b}_{x,y}| < |P^{-a,-b}_{x-a,y-b}| \Longleftrightarrow ax + by \leqslant 0; \tag{3.16}$$

$$\sigma_{i,j} \in P^{a,b}_{x,y} \Longrightarrow ai + bj < ax + by. \tag{3.17}$$

(3.14) and (3.15) are crucial for P to support optimal gossiping schemes respectively under the F* and the H* models. (3.17) implies that any GS based on P will gossip messages along shortest paths.

3.3 R^f for F* and R^h for H*

To design the round assignment R^h under the H* model, we need to schedule the edge orientation. If among $P^{a,b}_{x,y}$ σ is the i-th message arriving at $v_{x,y}$, then the

number i is denoted by $R^f_{x,y}(\sigma)$ and termed the arriving order of σ at $v_{x,y}$ (or in $P^{a,b}_{x,y}$). The relation between R^h and R^f reflects the edge orientation schedule. Intuitively, an edge should not stick to the same direction for a long time; otherwise some sending node might be exhausted. It seems success is more likely if we flip the edge as frequently as possible, especially in the first few rounds during which the nodes have not accumulated many messages. This intuition can be captured for $\sigma \in P^{a,b}_{x,y}$ as follows.

$$R^h_{x,y}(\sigma) = \begin{cases} 2R^f_{x,y}(\sigma) & ax+by \leqslant 0, \\ 2R^f_{x,y}(\sigma) - 1 & ax+by>0,\ R^f_{x,y}(\sigma) \leqslant \left|P^{-a,-b}_{x-a,y-b}\right|, \\ \left|P^{-a,-b}_{x-a,y-b}\right| + R^f_{x,y}(\sigma) & ax+by>0,\ R^f_{x,y}(\sigma) > \left|P^{-a,-b}_{x-a,y-b}\right|. \end{cases} \tag{3.18}$$

To understand this formulization, consider an edge $\{v_{x,y},\ v_{x-a,y-b}\}$. There are $|P^{a,b}_{x,y}|$ messages from $v_{x-a,y-b}$ to $v_{x,y}$ and $|P^{-a,-b}_{x-a,y-b}|$ messages from $v_{x,y}$ to $v_{x-a,y-b}$. By (3.16), $|P^{a,b}_{x,y}| < |P^{-a,-b}_{x-a,y-b}|$ if and only if $ax + by \leqslant 0$. So, if $|P^{a,b}_{x,y}| < |P^{-a,-b}_{x-a,y-b}|$, $v_{x,y}$ would receive from $v_{x-a,y-b}$ at the even rounds $2, 4, \ldots, 2|P^{a,b}_{x,y}| < n(n-1)/2$; otherwise, at the rounds $1,\ 3, \ldots,\ 2|P^{-a,-b}_{x-a,y-b}| - 1$ and the rounds $2|P^{-a,-b}_{x-a,y-a}|+1$ to $|P^{-a,-b}_{x-a,y-b}|+|P^{a,b}_{x,y}| = n(n-1)/2$. So on an edge, R^h starts with the majority direction (along which more messages travel), and reverses the direction at each round until the messages of the minority direction have all been transported. Thus, we do not need to worry about the initialized, 1-bounded, half-duplex, and optimal requirements for R^h.

(3.18) provides us an easier way for designing R^h, ordering the messages of each $P^{a,b}_{x,y}$ from 1 to $|P^{a,b}_{x,y}|$, to design an R^f that satisfies

$$\left\{R^f_{x,y}(\sigma)\middle|\sigma \in P^{a,b}_{x,y}\right\} = \left\{1, 2, \ldots, |P^{a,b}_{x,y}|\right\} \quad for\ (a,b) \neq (0,0). \tag{3.19}$$

For $(a,b) = (0,0)$, $R^f_{x,y}(\sigma_{x,y})$ is of course 0 so that to make $R^h_{x,y}(\sigma_{x,y})$ to be 0 for the initialization. Obviously, R^f could be an F* round assignment that pushes the messages of $P^{a,b}_{x,y}$ to $v_{x,y}$ one by one in round 1 to round $|P^{a,b}_{x,y}| \leqslant (n^2-1)/2$, if it is *precedence constrained* (as will see, it happens to be).

A message $\sigma \in P^{a,b}_{x,y}$ arriving at $v_{x,y}$ in round $R_{x,y}(\sigma) = n(n-1)/2$ is *dead* at $v_{x,y}$, with no further movement. By the discussion following (3.18), a sufficient and necessary condition for the existence of a dead message can be stated as

Lemma 3.1. *$P^{a,b}_{x,y}$ has a dead message if and only if $ax+by > 0$. If $ax+by > 0$, the dead message $\sigma \in P^{a,b}_{x,y}$ is the one with arriving order $R^f_{x,y}(\sigma) = |P^{a,b}_{x,y}|$.*

A message $\sigma \in P^{a,b}_{x,y}$ is said to be *hurried* into $v_{x,y}$ from $v_{x-a,y-b}$ if, after getting to $v_{x-a,y-b}$, it comes to $v_{x,y}$ at the first round when the edge takes the direction (a,b). For a message $\sigma \in P^{a,b}_{x-a,y-b}$, if σ is not dead at $v_{x-a,y-b}$ and comes to $v_{x-a,y-b}$ from $v_{x-2a,y-2b}$ with arriving order s, then, according to the edge orientation schedule (3.18), the earliest available order for it going to $v_{x,y}$ (being hurried) is

$$\alpha_{x,y}^{a,b}(s)=\begin{cases} s+1 & s\leqslant|P_{x-a,y-b}^{-a,-b}|,\\ 2s-|P_{x-a,y-b}^{-a,-b}| & |P_{x-a,y-b}^{-a,-b}|<s\leqslant|P_{x-2a,y-2b}^{-a,-b}|,\\ s+k+2 & |P_{x-2a,y-2b}^{-a,-b}|<s\leqslant|P_{x-a,y-b}^{a,b}|-1. \end{cases} \tag{3.20}$$

Moreover, all the messages $\sigma\in P_{x-a,y-b}^{0,0}\cup P_{x-a,y-b}^{a,b}$, except the one dead at $v_{x-a,y-b}$ (if it exists), can be hurried to $v_{x,y}$ with order $\alpha_{x,y}^{a,b}\big(R_{x-a,y-b}(\sigma)\big)$, causing neither contradiction with (3.19) nor conflict with the precedence constraint. Suppose we do so, among the $|P_{x,y}^{a,b}|$ orders of (3.19), we have used $|P_{x-a,y-b}^{a,b}|$ if $ax+by>1$ and $|P_{x-a,y-b}^{a,b}|+1$ otherwise.

By Lemma 3.1, $P_{x-a,y-b}^{a,b}$ has a dead message if and only if $ax+by>1$. If $ax+by\leqslant 1$, then $|P_{x-a,y-b}^{a,b}|\leqslant|P_{x-a,y-b}^{-a,-b}|$, leading to $\alpha_{x,y}^{a,b}\big(|P_{x-a,y-b}^{a,b}|\big)=|P_{x-a,y-b}^{a,b}|+1=|P_{x,y}^{a,b}|-k$, and implying that the k leftover orders are $|P_{x-a,y-b}^{a,b}|+t+1$ for $t=1,2,\ldots,k$. Otherwise, $|P_{x-a,y-b}^{a,b}|\geqslant|P_{x-a,y-b}^{-a,-b}|+2(k+1)$, and then from (3.20) we can see that for $s=|P_{x-a,y-b}^{-a,-b}|+t$ with $1\leqslant t\leqslant k+2$, $\alpha_{x,y}^{a,b}(s)=|P_{x-a,y-b}^{-a,-b}|+2t$, so the $k+1$ orders leftover should be $|P_{x-a,y-b}^{-a,-b}|+2t+1$ for $1\leqslant t\leqslant k+1$. For convenience, we define

$$\beta_{x,y}^{a,b}(t)=\begin{cases} |P_{x-a,y-b}^{a,b}|+t+1 & ax+by\leqslant 1,\\ |P_{x-a,y-b}^{-a,-b}|+2t+1 & ax+by>1. \end{cases} \tag{3.21}$$

$\beta_{x,y}^{a,b}$ specifies the orders not covered by $\alpha_{x,y}^{a,b}$. So, for (3.19), we have

Lemma 3.2. $\big\{1,2,\ldots,|P_{x,y}^{a,b}|\big\}=\big\{\beta_{x,y}^{a,b}(t)\big|1\leqslant t\leqslant k\big\}\cup\big\{\alpha_{x,y}^{a,b}(s)\big|0\leqslant s<|P_{x-a,y-b}^{a,b}|\big\}$

$$\cup\begin{cases} \big\{\alpha_{x,y}^{a,b}\big(|P_{x-a,y-b}^{a,b}|\big)\big\} & ax+by\leqslant 1,\\ \big\{\beta_{x,y}^{a,b}(k+1)\big\} & ax+by>1. \end{cases} \tag{3.22}$$

The proof is omitted. (3.22) suggests that for $\sigma\in P_{x,y}^{a,b}$ we define $R_{x,y}^{f}(\sigma)=\alpha_{x,y}^{a,b}\big(R_{x-a,y-b}^{f}(\sigma)\big)$ if $\sigma\in P_{x,y}^{0,0}\cup P_{x-a,y-b}^{a,b}$, and $\beta_{x,y}^{a,b}\big(t\big)$ otherwise. Actually, this suggestion needs to be fine tuned when used and not used for all cases. There are four cases to consider:

$$\begin{array}{ll} \textbf{Case 1:} & \sigma_{i,j}\in P_{x,y}^{0,1} \quad \text{where } -k\leqslant x,y<0;\\ \textbf{Case 2:} & \sigma_{i,j}\in P_{x,y}^{1,0} \quad \text{where } -k\leqslant x,y\leqslant 0;\\ \textbf{Case 3:} & \sigma_{i,j}\in P_{x,y}^{0,-1} \quad \text{where } -k\leqslant x\leqslant 0 \text{ and } -k\leqslant y<0;\\ \textbf{Case 4:} & \sigma_{i,j}\in P_{x,y}^{-1,0} \quad \text{where } -k\leqslant x\leqslant 0 \text{ and } -k\leqslant y<0. \end{array} \tag{3.23}$$

The other cases can be inferred from them by the rotating symmetry, $R_{x,y}^{f}(\sigma_{i,j})=R_{y,-x}^{f}(\sigma_{j,-i})$ (the lonely message $\sigma_{x,y}$ in $P_{x,y}^{0,0}$ has $R_{x,y}^{f}(\sigma_{x,y})=0$).

(3.22) is in fact the principle of "the first coming (to $v_{x-a,y-b}$), the first to leave (for $v_{x,y}$)" for handling $P_{x,y}^{a,b}\cap\big(P_{x-a,y-b}^{0,0}\cup P_{x-a,y-b}^{a,b}\big)$. Unfortunately, this principle does not work well for H*. It can be seen in Fig. 3 that at a node, while most of the messages will continue their journey along the direction they came, some of them will also make a turn for other destinations (productive), and some

will stop before reaching the mesh boundary (abortive). A reasonable idea is to hurry up those that will be *productive* at some later node, and slow down those that will be *abortive*. Our design of R^f for the first two cases reflects this idea. The suggestion (3.22) is adopted only in the last two cases where $ax + by > 0$.

Case 1: We need the messages in row x to reach $v_{x,y}$ early (hurried), and those in row $x+1$ to reach late (slowed). This can be done by letting $v_{x,y}$ receive $P^{0,1}_{x,y}$ in the order of rows $x, x-1, \ldots, -k, x+2, x+3, \ldots, 1, 0, x+1$. ($v_{-k,-k+1}$ is an exception, it receives in the order of rows $-k, -k+1, \ldots, 0$).

$$R^f_{x,y}(\sigma_{i,j})_{(\sigma_{i,j}\in P^{0,1}_{x,y})} = \begin{cases} i+k+1 & x{=}-k, y{=}-k{+}1, \\ (x-i)(k+y)+y-j & -k{\leqslant}i{\leqslant}x, \\ (k+i-1)(k+y)+y-j & x{+}1{<}i{\leqslant}0, \\ k(k+y)+y-j & i{=}x{+}1. \end{cases} \tag{3.24}$$

Case 2: We expect the messages of column y to come first, then the messages of column $k+y$, and those of column $y+1$ arrive last. This can not be realized immediately. The trick is to adjust $P^{1,0}_{-k+1,y}$ to $P^{1,0}_{0,y}$ little by little to achieve the aim at $v_{0,y}$.

$$R^f_{x,y}(\sigma_{i,j})_{(\sigma_{i,j}\in P^{1,0}_{x,y})} = \begin{cases} x-i & j{=}y, \\ (j-y)(k+x)+k+i+1 & (j{-}y)(k{+}x){+}i{<}0, \\ k+(k+y-j)(k+x)+x-i & (j{-}y)(k{+}x){+}i{\geqslant}0. \end{cases} \tag{3.25}$$

That is, $v_{x,y}$ first receives k messages in the order of column $y, y+1, y+2, \ldots$, and then it turns to receiving the leftover messages in the order of column $k+y, k+y-1, k+y-2 \ldots$. The mentioned intention is realized at $x = 0$.

Case 3: This is divided into two subcases: $y = -1$, and $y < -1$. We use function α to define $R^f_{x,y}$ recursively on $R^f_{x,y+1}$ (for $y+1 \geqslant 0$, $R^f_{x,y+1}$ is implied by applying rotating symmetry on (3.25), so the recursion is well defined).

$$R^f_{x,y}(\sigma_{i,j})_{(\sigma_{i,j}\in P^{0,-1}_{x,y})(y=-1)} = \begin{cases} \alpha^{0,-1}_{x,y}\big(R^f_{x,y+1}(\sigma_{i,j})\big) & \sigma_{i,j}\in P^{0,0}_{x,y+1}\cup P^{0,-1}_{x,y+1}, \\ \alpha^{0,-1}_{x,y}\big(R^f_{x,y+1}(\sigma_{x+1,\ -i+1})\big) & x{<}i{\leqslant}0,\ j{=}y{+}1, \\ \alpha^{0,-1}_{x,y}\big(R^f_{x,y+1}(\sigma_{x+1,\ -i})\big) & -k{\leqslant}i{<}x,\ j{=}k{+}y{+}1, \\ \beta^{0,-1}_{x,y}(i) & 0{<}i{\leqslant}k,\ j{=}y{+}1. \end{cases} \tag{3.26}$$

$$R^f_{x,y}(\sigma_{i,j})_{(\sigma_{i,j}\in P^{0,-1}_{x,y})(y<-1)} = \begin{cases} \alpha^{0,-1}_{x,y}\big(R^f_{x,y+1}(\sigma_{i,j})\big) & \sigma_{i,j}\in P^{0,0}_{x,y+1}\cup P^{0,-1}_{x,y+1}, \\ \beta^{0,-1}_{x,y}(-i+2) & x{<}i{\leqslant}0,\ j{=}y{+}1, \\ \beta^{0,-1}_{x,y}(-i+1) & -k{\leqslant}i{<}x,\ j{=}k{+}y{+}1, \\ \beta^{0,-1}_{x,y}(1) & i{=}k{+}y{+}2,\ j{=}y{+}1, \\ \alpha^{0,-1}_{x,y}\big(R^f_{x,y+1}(\sigma_{i,\ y+2})\big) & 0{<}i{\leqslant}k,\ i{\neq}k{+}y{+}2\ j{=}y{+}1. \end{cases} \tag{3.27}$$

Case 4: This is divided into three subcases: $x = 0$, $x = -1$, and $x < -1$. At $x = 0$, it is similar to (3.24) except that the messages in column $y+1$ are swapped with those of $P^{0,-1}_{x,y}$ in row $y+1$. For $x < 0$, it is similar to Case 3 but trickier in assigning the rounds to the messages in row $x+1$ and $k+x+1$. This is because, for large n, when y is approaching $-k$, the messages of row $k+y+1$ come (from right) to $v_{x+1,y}$ too late.

$$R^f_{x,y}(\sigma_{i,j})\ (\sigma_{i,j}\in P^{-1,0}_{x,y})\ (x{=}0) = \begin{cases} (y-j)(k-x)+i-x & -k\leqslant j\leqslant y,\\ (k+j-1)(k-x)+i-x & y{+}1<j\leqslant 0,\\ k(k-x)+j & i{=}x{+}1,\ 0<j\leqslant k. \end{cases} \tag{3.28}$$

$$R^f_{x,y}(\sigma_{i,j})\ (\sigma_{i,j}\in P^{-1,0}_{x,y})\ (x{=}{-}1) = \begin{cases} \alpha^{-1,0}_{x,y}\big(R^f_{x+1,y}(\sigma_{i,j})\big) & \sigma_{i,j}\in P^{0,0}_{x+1,y}\cup P^{-1,0}_{x+1,y},\\ \alpha^{-1,0}_{x,y}\big(R^f_{x+1,y}(\sigma_{x+2,j+1})\big) & i{=}k{+}x{+}1,\ 0<j<k,\\ \beta^{-1,0}_{x,y}(1) & i{=}x{+}k{+}1,\ j{=}k,\\ \alpha^{-1,0}_{x,y}\big(R^f_{x+1,y}(\sigma_{x+2,j})\big) & i{=}x{+}1,\ j{=}1,\\ \beta^{-1,0}_{x,y}(j+1) & i{=}x{+}1,\ 1<j\leqslant k. \end{cases} \tag{3.29}$$

$$R^f_{x,y}(\sigma_{i,j})\ (\sigma_{i,j}\in P^{-1,0}_{x,y})\ (x{<}{-}1) = \begin{cases} \alpha^{-1,0}_{x,y}\big(R^f_{x+1,y}(\sigma_{i,j})\big) & \sigma_{i,j}\in P^{0,0}_{x+1,y}\cup P^{-1,0}_{x+1,y},\\ \alpha^{-1,0}_{x,y}\big(R^f_{x+1,y}(\sigma_{x+2,j})\big) & i{=}x{+}1,\ 0<j\leqslant k{+}x{+}1,\\ \beta^{-1,0}_{x,y}(j-k-x-1) & i{=}x{+}1,\ k{+}x{+}1<j\leqslant k,\\ \beta^{-1,0}_{x,y}(j-x-1) & i{=}k{+}x{+}1,\ 0<j\leqslant k{+}x{+}2,\\ \alpha^{-1,0}_{x,y}\big(R^f_{x+1,y}(\sigma_{x+2,j})\big) & i{=}k{+}x{+}1,\ k{+}x{+}2<j\leqslant k. \end{cases} \tag{3.30}$$

◎ 01 02 03 04 05 06	01 ◎ 01 02 03 04 05	02 01 ◎ 01 02 03 04	03 02 01 ◎ 01 02 03
01 13 22 23 **19 07 09**	02 **01** 17 18 **19 07 09**	08 07 **01** 13 **19 07 09**	**16 07 24 01** 10 11 12
02 11 20 21 14 15 17	03 **02** 15 16 09 10 12	04 03 **02** 12 08 09 10	06 05 04 **02** 07 08 09
03 09 18 19 08 10 12	04 **03** 13 14 06 07 08	06 05 **03** 11 05 06 07	09 08 07 **03** 04 05 06
04 24 **08 14 11 13 16**	**08 04** 19 **14 11 13 16**	**14 08 04** 14 **11 13 16**	**08 14** 10 **04 09 11 13**
05 07 **10 15 18 20 22**	**10 05** 20 **15 18 20 22**	**15 10 05** 15 **18 20 22**	**10 15** 11 **05 18 20 22**
06 16 **12 17 21 23 24**	**12 06** 11 **17 21 23 24**	**17 12 06** 16 **21 23 24**	**12 17** 12 **06 19 21 23**
01 02 03 04 13 22 23	02 01 02 03 04 17 18	04 03 01 02 03 04 13	06 05 04 01 02 03 04
◎ 01 02 03 04 05 06	01 ◎ 01 02 03 04 05	02 01 ◎ 01 02 03 04	03 02 01 ◎ 01 02 03
01 11 20 21 **14 20 11**	04 **01** 15 16 **14 20 11**	08 07 **01** 12 **14 20 11**	**11 20 19 01** 10 11 12
02 09 18 19 14 15 17	03 **02** 13 14 09 10 12	06 05 **02** 11 08 09 10	09 08 07 **02** 07 08 09
03 24 **06 09** 08 10 12	**06 03** 19 **09** 06 07 08	**09 06 03** 14 05 06 07	**06 09** 10 **03** 04 05 06
04 07 **07 10 13 15 17**	**07 04** 20 **10 13 15 17**	**10 07 04** 15 **13 15 17**	**07 10** 11 **04 13 15 17**
05 16 **08 12 16 18 19**	**08 05** 11 **12 16 18 19**	**12 08 05** 16 **16 18 19**	**08 12** 12 **05 14 16 18**
02 03 07 05 13 22 23	03 02 03 07 05 17 18	06 05 02 03 07 05 13	09 08 07 02 03 07 05
01 08 06 04 11 20 21	02 01 08 06 04 15 16	04 03 01 08 06 04 12	06 05 04 01 08 06 04
◎ 01 02 03 04 05 06	01 ◎ 01 02 03 02 05	02 01 ◎ 01 02 03 04	03 02 01 ◎ 01 02 03
01 09 18 19 **11 15 16**	04 **01** 13 14 **11 15 16**	08 07 **01** 11 **11 15 16**	**14 15 16 01** 10 11 12
02 24 **05 08** 14 15 17	**05 02** 19 **08** 09 10 12	**08 05 02** 14 08 09 10	**05 08** 10 **02** 07 08 09
03 07 **06 09** 08 10 12	**06 03** 20 **09** 06 07 08	**09 06 03** 15 05 06 07	**06 09** 11 **03** 04 05 06
04 16 **07 10 12 13 14**	**07 04** 11 **10 12 13 14**	**10 07 04** 16 **12 13 14**	**07 10** 12 **04 11 12 13**
03 12 09 06 13 22 23	04 03 12 09 06 17 18	05 04 03 12 09 06 13	06 05 04 03 12 09 06
02 11 08 05 11 20 21	03 02 11 08 05 15 16	07 06 02 11 08 05 12	09 08 07 02 11 08 05
01 10 07 04 09 18 19	02 01 10 07 04 13 14	03 08 01 10 07 04 11	12 11 10 01 10 07 04
◎ 01 02 03 04 05 06	01 ◎ 01 02 03 04 05	02 01 ◎ 01 02 03 04	03 02 01 ◎ 01 02 03
01 24 **04 07 10 11 12**	**04 01** 19 **07 10 11 12**	**07 04 01** 14 **10 11 12**	**04 07 10 01** 10 11 12
02 07 **05 08** 14 15 17	**05 02** 20 **08** 09 10 12	**08 05 02** 15 08 09 10	**05 08 11 02** 07 08 09
03 16 **06 09** 08 10 12	**06 03** 11 **09** 06 07 08	**09 06 03** 16 05 06 07	**06 09 12 03** 04 05 06

Fig. 5. $R^f_{x,y}(\sigma_{i,j})$ in $M_{7\times 7}$ for $-3\leqslant x,y\leqslant 0$

The design of R^f is finished, and so is R^h. As examples in $M_{7\times 7}$, Fig. 5 shows the values of $R^f_{x,y}(\sigma_{i,j})$ and Fig. 6 the values of $R^h_{x,y}(\sigma_{i,j})$. We conclude with the next theorem (proof omitted).

Theorem 3.3. *(P,R^f) is an $F^*\big((n^2-1)/2\big)$-GS and (P,R^h) is an $H^*\big(n(n+1)/2\big)$-GS, both optimal and routing messages along shortest paths in $M_{n\times n}$ with odd $n>1$.*

◎ 01 03 05 07 09 10	02 ◎ 01 03 05 07 09	04 02 ◎ 01 03 05 07	06 04 02 ◎ 02 04 06
01 17 26 27 **23 11 13**	04 **01** 25 26 **23 11 13**	16 14 **01** 25 **23 11 13**	**20 11 28 01** 20 22 24
03 15 24 25 18 19 21	06 **03** 23 24 17 18 20	08 06 **03** 23 15 17 19	12 10 08 **03** 14 16 18
05 13 22 23 12 14 16	08 **05** 21 22 11 13 15	12 10 **05** 21 09 11 13	18 16 14 **05** 08 10 12
07 28 **12 18 15 17 20**	**12 07** 27 **18 15 17 20**	**18 12 07** 26 **15 17 20**	**12 18** 20 **07 13 15 17**
09 11 **14 19 22 24 26**	**14 09** 28 **19 22 24 26**	**19 14 09** 27 **22 24 26**	**14 19** 22 **09 22 24 26**
10 20 **16 21 25 27 28**	**16 10** 19 **21 25 27 28**	**21 16 10** 28 **25 27 28**	**16 21** 24 **10 23 25 27**
02 04 06 08 17 26 27	04 02 04 06 08 25 26	08 06 02 04 06 08 25	12 10 08 02 04 06 08
◎ 01 03 05 07 09 10	02 ◎ 01 03 05 07 09	04 02 ◎ 01 03 05 07	06 04 02 ◎ 02 04 06
01 15 24 25 **22 28 19**	08 **01** 23 24 **22 28 19**	16 14 **01** 23 **22 28 19**	**19 28 27 01** 20 22 24
03 13 22 23 18 19 21	06 **03** 21 22 17 18 20	12 10 **03** 21 15 17 19	18 16 14 **03** 14 16 18
05 28 **11 17** 12 14 16	**11 05** 27 **17** 11 13 15	**17 11 05** 26 09 11 13	**11 17** 20 **05** 08 10 12
07 11 **13 18 21 23 25**	**13 07** 28 **18 21 23 25**	**18 13 07** 27 **21 23 25**	**13 18** 22 **07 21 23 25**
09 20 **15 20 24 26 27**	**15 09** 19 **20 24 26 27**	**20 15 09** 28 **24 26 27**	**15 20** 24 **09 22 24 26**
04 06 14 10 17 26 27	06 04 06 14 10 25 26	12 10 04 06 14 10 25	18 16 14 04 06 14 10
02 16 12 08 15 24 25	04 02 16 12 08 23 24	08 06 02 16 12 08 23	12 10 08 02 16 12 08
◎ 01 03 05 07 09 10	02 ◎ 01 03 05 07 09	04 02 ◎ 01 03 05 07	06 04 02 ◎ 02 04 06
01 13 22 23 **21 27 28**	08 **01** 21 22 **21 27 28**	16 14 **01** 21 **21 27 28**	**26 27 28 01** 20 22 24
03 28 **09 15** 18 19 21	**09 03** 27 **15** 17 18 20	**15 09 03** 26 15 17 19	**09 15** 20 **03** 14 16 18
05 11 **11 17** 12 14 16	**11 05** 28 **17** 11 13 15	**17 11 05** 27 09 11 13	**11 17** 22 **05** 08 10 12
07 20 **13 19 23 25 26**	**13 07** 19 **19 23 25 26**	**19 13 07** 28 **23 25 26**	**13 19** 24 **07 21 23 25**
06 24 18 12 17 26 27	08 06 24 18 12 25 26	10 08 06 24 18 12 25	12 10 08 06 24 18 12
04 22 16 10 15 24 25	06 04 22 16 10 23 24	14 12 04 22 16 10 23	18 16 14 04 22 16 10
02 20 14 08 13 22 23	04 02 20 14 08 21 22	06 16 02 20 14 08 21	24 22 20 02 20 14 08
◎ 01 03 05 07 09 10	02 ◎ 01 03 05 07 09	04 02 ◎ 01 03 05 07	06 04 02 ◎ 02 04 06
02 28 **08 14 20 22 24**	**08 02** 27 **14 20 22 24**	**14 08 02** 26 **20 22 24**	**08 14 20 02** 20 22 24
04 11 **10 16** 18 19 21	**10 04** 28 **16** 17 18 20	**16 10 04** 27 15 17 19	**10 16 22 04** 14 16 18
06 20 **12 18** 12 14 16	**12 06** 19 **18** 11 13 15	**18 12 06** 28 09 11 13	**12 18 24 06** 08 10 12

Fig. 6. $R^h_{x,y}$ in $M_{7\times 7}$ for $-3 \leqslant x, y \leqslant 0$

4 Even n

The same ideas for odd n applies as well to even $n(=2k)$. For space reason, we state the next theorem without proof.

Theorem 4.1. *There are an F^* $\lceil (n^2-1)/2 \rceil$-GS and an H^* $(n(n+1)/2)$-GS, both of which route messages along shortest paths in $M_{n\times n}$ of even n.*

As in the odd n case, both schemes are rotating symmetric and based on the same P. For $-k \leqslant x, y \leqslant -1$, $P^{a,b}_{x,y}$ is illustrated in Fig. 7, and formulated below. Note that, for even n, the subscripts of P or σ cannot be 0. We use the new operators $\boxplus$ and $\oplus$ in place of $+$ to take the values -1 and 1, respectively, once 0 is produced. For example, when $x = -k$, $x \boxplus k = -1$ and $x \oplus k = 1$.

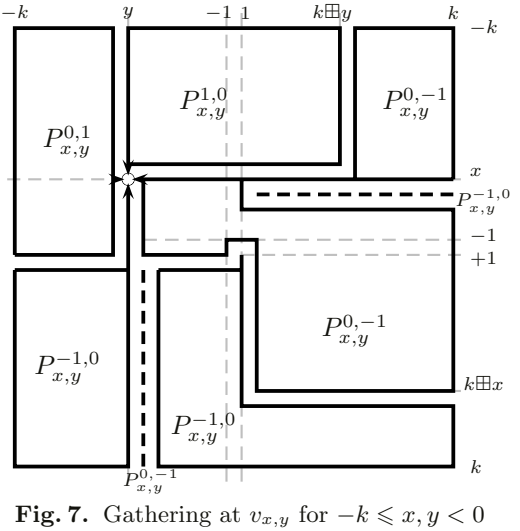


Fig. 7. Gathering at $v_{x,y}$ for $-k \leqslant x, y < 0$

$$P^{0,0}_{x,y} = \{\sigma_{x,y}\}; \tag{4.1}$$

$$P^{0,1}_{x,y} = \{\sigma_{i,j}|_{-k\leqslant i\leqslant 1,\ -k\leqslant j<y}\}; \tag{4.2}$$

$$P^{1,0}_{x,y} = \{\sigma_{i,j}|_{-k\leqslant i<x,\ y\leqslant j\leqslant k\boxplus y}\}; \tag{4.3}$$

$$\begin{aligned} P^{0,-1}_{x,y} &= P^{0,0}_{x,y\oplus 1} \cup P^{0,-1}_{x,y\oplus 1} \cup \{\sigma_{i,k+y+1}|_{-k\leqslant i<x}\} \cup \{\sigma_{i,y\oplus 1}|_{1<i\leqslant k}\} \\ &\quad \cup \begin{cases} \{\sigma_{i,y\oplus 1}|_{x<i<1}\} - \{\sigma_{x\oplus 1,j}|_{1<j\leqslant k}\} & y=-1, \\ \{\sigma_{i,y\oplus 1}|_{x<i\leqslant 1}\} - \{\sigma_{i,y\oplus 2}|_{1<i\leqslant k}\} & y<-1; \end{cases} \end{aligned} \tag{4.4}$$

$$\begin{aligned} P^{-1,0}_{x,y} &= P^{0,0}_{x\oplus 1,y} \cup P^{-1,0}_{x\oplus 1,y} \cup \{\sigma_{k+x+1,j}|_{1<j\leqslant k}\} \\ &\quad \cup \{\sigma_{x\oplus 1,j}|_{1<j\leqslant k}\} - \begin{cases} \{\sigma_{i,y\oplus 1}|_{1<i\leqslant k}\} \cup \{\sigma_{1,1}\} & x=-1, \\ \{\sigma_{x\oplus 2,j}|_{1<j\leqslant k}\} & x<-1. \end{cases} \end{aligned} \tag{4.5}$$

Clearly, P is complete and duplication free. For $(a,b) \neq (0,0)$, there are some facts about $|P^{a,b}_{x,y}|$, which are similar to those of (3.14–3.16). For examples, $|P^{a,b}_{x,y}| + |P^{-a,-b}_{x-a,y-b}| = n(n+1)/2$, and

$$|P^{a,b}_{x,y}| = \left\{ \begin{array}{ll} (k+ax+by)(k+1) & ax+by<1,\ ay-bx\geqslant 1; \\ k(k+1) & ax+by=1,\ ay-bx\geqslant 1; \\ (k+ax+by)k & ax+by>1,\ ay-bx\geqslant 1; \\ (k+ax+by)k & ax+by<1,\ ay-bx<1; \\ k^2 & ax+by=1,\ ay-bx<1; \\ (k+ax+by)k+ax+by-k-1 & ax+by>1,\ ay-bx<1. \end{array} \right\} \leqslant \left\lceil \frac{n^2-1}{2} \right\rceil.$$

The above are necessary for schemes based on P to match the lower bounds. R^f and R^h are omitted because of space. They can be defined similarly as is done in Section 3.

5 Conclusion and Discussion

This paper mainly focuses on square meshes and presents optimal gossiping algorithms under the F* and H* models with $p = 1$. As mentioned at the end of Section 3.1, the F* alogrithm can be generalized to gossip optimally in nonsquare meshes. Thus the problems about meshes are solved with the positive answers that $g_{F*}(M_{m\times n}) = \lceil (mn-1)/2 \rceil$ and $g_{H*}(M_{n\times n}) = n(n+1)/2$. It seems possible that the same ideas can be applied to higher dimensional square meshes and tori to obtain optimal or near-optimal results. For non-square 2D meshes, relaxing the rotating symmetry from 90° to 180° (e.g., $P^{a,b}_{x,y} = P^{-a,-b}_{-x,-y}$), the method can be extended to result in fast H* algorithms.

Using some vector *language*, the solutions can be expressed in a more unified way, and the proofs can be conducted succinctly. Because of the lack of space, we omitted many of the proofs which can be found in an unpublished manuscript at www.cs.hku.hk/∼fcmlau/gossip.pdf.

Acknowledgement

This work is supported in part by a Hong Kong RGC grant ("Fast gossiping for mesh-connected parallel computers").

References

1. A. Bagchi, E.F. Schmeichel, and S.L. Hakimi, "Sequential information dissemination by packets", *Networks*, vol. 22, pp. 317–333, 1992.
2. A. Bagchi, E.F. Schmeichel, and S.L. Hakimi, "Parallel information dissemination by packets", *SIAM J. Computing*, vol. 23, pp. 355–372, 1994.
3. J.-C. Bermond, L. Gargano, A.A. Rescigno and U. Vaccaro, "Fast gossiping by short messages", *Proc. Int'l Colloquium Automata, Languages, and Processing'95*, pp. 135–146, 1995.
4. M. Chrobak, L. Gasieniec, and W. Rytter, "Fast broadcasting and gossiping in radio networks", *Proc. of 41st Annual Symposium on Foundations of Computer Science*, 2000, pp. 575–581.
5. J.-C. Bermond, L. Gargano, and S. Perennes, "Optimal sequential gossiping by short messages", *Discrete Applied Mathematics*, vol. 86, pp. 145–155, 1998.
6. W.J. Dally, and P. Song, "Design of self-timed VLSI multicomputer communication controller", *Proc. Int'l Conf. Computer Design*, pp. 230–234, 1987.
7. P. Fraigniaud and E. Lizard, "Methods and problems of communication in usual networks", *Discrete Applied Math.*, vol. 53, pp. 79–134, 1994.
8. S. Fujita and M. Yamashita, "Fast gossiping on square mesh computers", *Information Processing Letters*, vol. 48, pp. 127–130, 1993.
9. S.M. Hedetniemi, S.T. Hedetniemi, and A. Liestman, "A survey of gossiping and broadcasting in communication networks", *Networks*, vol. 18, pp. 319–349, 1988.
10. J. Hromkovic, R. Klasing, B. Monien and R. Peine, "Dissemination of information in interconnection networks (Broadcasting & Gossiping)", *Combinatorial Network Theory*, pp. 125–212, D.-Z. Du and D.F. Hsu, eds., 1996.
11. D.W. Krumme, "Fast gossiping for the hypercube", *SIAM J. Computing*, vol. 21, no.2, pp. 365–380, Apr. 1992.
12. F.C.M. Lau and S.H. Zhang, "Fast gossiping in square meshes/tori with bounded-size packets", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, No. 4, April, 2002, pp. 349–358.
13. F.C.M. Lau and S.H. Zhang, "Optimal gossiping in paths and cycles", *J. of discrete algorithms*, vol. 1, no. 5-6, pp. 461–475, Oct. 2003.
14. M.D. May, P.W. Thompson, and P.H. Welch, "Networks, routers, and transputers", *Amsterdam: IOS Press*, 1993.
15. M. Soch and P. Tvrdik, "Optimal gossiping in noncombining 2D meshes", *Proc. Int'l Colloquium Structural Information Comm. Complexity* (SIROCCO'97), 1997.

Geometric Routing Without Geometry

Mirjam Wattenhofer[1], Roger Wattenhofer[2], and Peter Widmayer[1]

[1] Department of Computer Science, ETH Zurich
[2] Computer Engineering and Networks Laboratory, ETH Zurich

Abstract. In this paper we propose a new routing paradigm, called *pseudo-geometric routing*. In pseudo-geometric routing, each node u of a network of computing elements is assigned a *pseudo coordinate* composed of the graph (hop) distances from u to a set of designated nodes (the *anchors*) in the network. On theses pseudo coordinates we employ greedy geometric routing. Almost as a side effect, pseudo-geometric routing is not restricted to planar unit disk graph networks anymore, but succeeds on general networks.

1 Introduction

With the advent of ad hoc, sensor, mesh or peer-to-peer systems, networking research has recently received a second wind of attention. In the center of interest is the routing problem. Contrary to established networks such as the Internet, these new networks ask for a novel generation of routing protocols. First, ad hoc or sensor networks run on plain and feeble hardware that does not allow nodes to store large routing tables as needed by classic routing algorithms such as distance-vector or link-state routing. Second, peer-to-peer or ad hoc networks are highly dynamic – the topology of the network is changing constantly, at much higher rates than in conventional networks such as the Internet. Consequently, this leads to an immense exchange of control messages in classic routing protocols.

In order to tackle the routing problem for ad hoc/sensor/mesh/peer-to-peer networks, the research community has proposed an array of innovative routing protocols and paradigms. Originated from multihop radio networks, geometric routing is a prominent representative, combining small memory overhead with few updates per topology change. In geometric routing the nodes do not have routing tables at all, instead it is assumed that the nodes have *coordinates* in the Euclidean plane.

The early proposals of geometric routing—suggested twenty years ago by Takagi and Kleinrock [25]—were of purely greedy nature: Each node knows its own coordinate, as well as the coordinates of its neighbors. When receiving a message containing the coordinate of the destination, a node forwards the message to its "best" neighbor – the neighbor node geometrically closest to the destination.

Yet, already in simple configurations greedy geo-routing can fail if the message reaches a local minimum with respect to the distance to the destination, that is a node without any "better" neighbors. This deadlock, however, can be resolved by using more elaborate geo-routing protocols (see Section 2). In this paper we advocate using the original greedy geo-routing, however, with a higher-dimensional geometric space.

A. Pelc and M. Raynal (Eds.): SIROCCO 2005, LNCS 3499, pp. 307–322, 2005.

Another problem is the availability of position information (coordinates) which is needed to run a geo-routing algorithm. Clearly, one possible technical solution is to equip each node with a Global Positioning System (GPS) receiver. However, in comparison to a sensor node, a GPS receiver is clumsy, expensive, and energy-inefficient. Moreover, GPS reception might be obstructed by climatic conditions; if nodes are deployed indoors, there is no reception at all.

As a GPS-alternative, researchers proposed to compute so-called *virtual coordinates* merely from connectivity or distance information and employ geometric routing schemes on those coordinates. However, the apparent computational complexity of virtual coordinates [20, 18] discourages from using them in real systems.

In this paper we advocate a paradigm shift. Instead of trying to solve the tough virtual coordinates problem and then use advanced geo-routing techniques, we go back to the roots and use greedy geometric routing on down-to-earth virtual coordinates.

In particular, we propose that the virtual (or *pseudo*) coordinate of a node u is a vector, composed by the graph (hop) distances from u to a set of designated nodes (the *anchors*) in the network. Almost as a side effect, pseudo-geometric routing is not restricted to planar unit disk graph networks anymore, but succeeds on general networks. We believe that the coordinate of a node will be relatively stable even if the network topology changes, hence making our routing scheme applicable for highly dynamic networks.

In order to gain a deeper insight into this new routing paradigm and to explore its algorithmic foundations and limits, we assess its potential by investigating various basic network topologies.

After giving an overview of related work in the following section, we state the model used in this paper in Section 3. In Sections 4 – 9 we analyze the properties of our routing paradigm for different network topologies, namely rings, trees, grids, unit disk graphs, butterfly networks, and hypercubes. Section 10 concludes on the paper.

2 Related Work

There are a hand full of routing protocols which are similar in spirit. We discuss selected protocols in this section.

The link-reversal paradigm [11] improves significantly on the standard distance-vector routing protocol[1] by not updating the distances with each topology change. Recently, the performance of the link-reversal paradigm was analyzed [4]. Besides being more apt than distance-vector routing to be applicable in highly dynamic networks, the link-reversal protocol also requires less memory.

Speaking of memory-efficiency: In classic large scale communication networks a dominant problem is to develop *compact* routing schemes which feature low memory overhead per node and still produce efficient routes between source and destination. The first routing scheme which addresses the efficiency-memory tradeoff was proposed

[1] In distance-vector, each node stores the distance to each destination, and which link to follow – a derivative of distance-vector is deployed in the Internet BGP protocol.

in [16], where the idea of hierarchically clustering a network into levels and using the resulting structure for routing was introduced.

Subsequently, the trade-off between memory space and stretch factor was theoretically analyzed [21, 14] and a plethora of compact routing schemes was proposed [8, 5, 26, 2]. For comprehensive surveys on compact routing see [12, 13, 15].

An important branch of compact routing, so called *interval* routing, is based on the idea of grouping nodes in cyclic intervals and was first suggested in [23] for tree networks. Later on this work was extended to other network topologies [27, 9, 7]. It is worth noting that an interval routing scheme, once computed for a graph, can be used to perform other tasks than routing. [10] proposed a $\Theta(n)$ broadcast algorithm that uses only interval routing labels.

The best compact routing schemes provide amazing memory-stretch ratios. However, they are hardly applicable in dynamic networks since all routing tables have to be computed from scratch if the topology of the network changes.

A well-studied routing paradigm for radio networks is geometric (a.k.a. geographic, location-based, position-based, or simply geo-routing) routing. The first proposals were of purely greedy nature. As pointed out in the introduction, already in simple configurations greedy geo-routing can fail if a message reaches a local minimum. This deadlock problem, however, was resolved by the employment of face routing, which explores the boundaries of faces of the planarized network graph [17]. In recent years, geo-routing has experienced several improvements. The routing schemes GFG [3] and GOAFR+ [19] advocate a combination of greedy and face routing. Whereas GFG does not give competitive worst-case guarantees, GOAFR+ is a routing algorithm which is efficient for average-case networks as well as asymptotically worst-case optimal. Recently, the locality aware location service LLS [1] proposes a solution how the coordinates of mobile destinations can be learned efficiently using a peer-to-peer-like scheme.

Unfortunately, it is not always feasible to assume that each node in the network knows its position. As an alternative, researchers proposed to compute so-called *virtual coordinates*, coordinates computed merely from connectivity or distance information, and employ geo-routing schemes on top of these coordinates. In [22] a greedy routing scheme is employed on the virtual coordinates which are obtained by a spring-based algorithm. By solving a convex linear program the coordinates of the network nodes are estimated in [6], whereas the heuristic in [24] is based on multidimensional scaling.

Apart from these heuristics there is little work on virtual coordinates: In [20] the authors present an approximation algorithm for the virtual coordinates problem, with polylogarithmic approximation ratio only. In a lower bound paper [18] it is shown that virtual coordinates cannot be approximated arbitrarily well. These two results dampen our hopes that using geo-routing on virtual coordinates in real systems is practical.

Instead of embedding the nodes in the two dimensional Euclidean space, in this paper we propose to embed the nodes in a sufficiently high dimensional pseudo geometric space and employ a greedy geo-routing scheme on top.

3 Model

Let a network on n nodes be given, where k nodes $a_1, a_2, \ldots, a_k$ are designated *anchors* and there exists a unique order on the anchors with $a_1 \prec a_2 \prec \ldots \prec a_k$. Each node u in the networks knows the underlying network topology and is furthermore able to determine its graph (hop) distance d_i to each one of the k anchors a_i. The *(pseudo-) coordinate* of node u is then defined to be $(d_1, \ldots, d_k)$. Thus, the network is embedded in a pseudo k-dimensional space. Each node knows in addition to its own coordinate the coordinates of all its direct neighbors.

In *pseudo-geometric routing algorithms* when receiving a message containing the (pseudo) coordinate of the destination, a node forwards the message to its "best" neighbor – the neighbor node geometrically closest (in the pseudo geometric space) to the destination. In the following we say that the pseudo-geometric routing problem can be solved if there is a pseudo-geometric routing algorithm which guarantees message delivery.

In general, a routing algorithm may have two basic problems. The first and foremost problem, we henceforth also call *naming problem*, is that all nodes must have unique identifiers, otherwise the destination is not identifiable in general. Once the naming problem is solved, the algorithm furthermore has to guarantee that any destination must finally be reached from any source –the *routing* problem.

In the following, we exemplarily demonstrate at the showcase where the graph is a line which properties of the pseudo-geometric routing problem we analyze, why these properties are important from a theoretical and practical point of view and how the next sections are structured. In general, we first concentrate on the naming problem before we finally give a pseudo-geometric routing scheme.

The first property we explore is the *minimal* number of anchors we need to solve the naming problem. Theoretically speaking, we give a *lower bound* on the number of anchors. From a practical point of view this is a quite natural property. Amongst others, it gives the minimal amount of storage which is needed per node to solve the naming problem.

Example: *If a node* a *with degree one on the line is chosen to be an anchor, each node on the line has a unique coordinate, since each node has a unique distance to* a*. This leads to the following lemma.*

Lemma 1 (min). *Choosing a node on the line with degree one solves the naming problem.*

The second property we are interested in is an *upper bound* on the number of anchors which are needed to solve the naming problem, in the following sense. If we allow an adversary to choose the anchors arbitrarily, how many anchors must be chosen until the naming problem is solved? In practice it is often not feasible to deliberately assign anchors, but anchors are chosen more or less arbitrarily, hence we might come across this upper bound.

Example: *For the line, this upper bound is trivially obtained by observing that for any two nodes on the line each node has a unique distance vector.*

Lemma 2 (any). *Two arbitrarily chosen anchors on a line solve the naming problem.*

In networks another crucial factor is the degree of *locality*. Having a local solution is clearly favorable to having a global solution, where we use local in the sense of the anchors being in a constant size neighborhood of each other. The advantage of local solutions lies in the fact that failures and updates can be dealt with locally. Leading to the third property we look at our example.

Example: *Since two arbitrary anchors on the line solve the naming problem, clearly two* incident *anchors solve the naming problem.*

Lemma 3 (local). *Two* incident *nodes on the line solve the naming problem.*

Finally, we give a pseudo-geometric routing scheme. Towards this goal we explicitly assume that the anchors are chosen in some way and based on this choice of anchors show how nodes pass a message such that it eventually reaches the destination. For most of the analyzed topologies we show that the chosen path is actually the shortest path between source and destination.

Example: *Choose one anchor on the line, namely a node with degree one. Based on its own coordinate and the coordinate of the destination, a node immediately knows to which neighbor to pass the message such that it reaches the destination on the shortest path.*

Theorem 4. *The pseudo-geometric routing problem on the line can be solved (locally) with one anchor.*

Wrapping up, in the following sections we concentrate for each topology on three important properties related to the naming problem. The first property called *min* is a lower bound on the number of anchors we need to solve the naming problem, whereas *any* is an upper bound. *Local* gives a choice of anchors which solves the naming problem and is local, if such a choice exists. With local we mean that the anchors are within constant graph distance from each other. We then give a pseudo-geometric routing scheme which guarantees message delivery.

4 Ring

4.1 Naming

With one anchor a only, there are $\lfloor (n-1)/2 \rfloor$ pairs of nodes with pairwise same distance to a, that is, they cannot be distinguished. On the other hand, with two anchors a and b, which are at distance $d \neq n/2$ to each other, or arbitrary three anchors each node has a unique coordinate. Leading to following lemmas.

Lemma 5 (min, local). *If chosen properly, exactly 2 anchors solve the naming problem in the ring, specifically two* incident *anchors solve the naming problem.*

Lemma 6 (any). *3 arbitrarily chosen anchors solve the naming problem in the ring.*

4.2 Routing

Suppose we have two anchors which solve the naming problem in the ring. A node u can reconstruct the position of the anchors based on its and its neighbors' coordinates. Furthermore, u knows the position of the destination relative to the anchors and hence can pass the message to the neighbor which is nearest to the destination. By this discussion and Lemma 5 the following theorem can be deduced immediately.

Theorem 7. *The pseudo-geometric routing problem on the ring can be solved (locally) with 2 anchors. Furthermore, the chosen route between source and destination is a shortest path.*

5 Grid

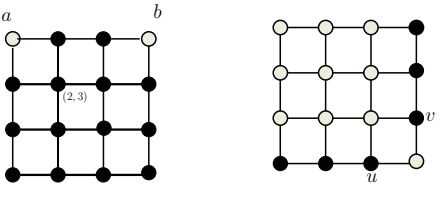
5.1 Naming

With one anchor only at least $\sqrt{n}$ nodes do have the same coordinate in the grid. On the other hand, if we choose one anchor a such that it lies in the upper left corner of the grid and another anchor b which lies in the upper right corner of the grid, all nodes have different coordinates (see Figure 1(a)).

Lemma 8 (min). *If chosen properly, we need exactly 2 anchors in the grid to solve the naming problem.*

Lemma 9 (local). *It is not possible to solve the naming problem in the grid locally.*

Proof. Consider an arbitrary subgraph of the grid with constant diameter, where all nodes in the subgraph are anchors. Then there are two nodes incident to a corner node which are not anchors. Those two nodes cannot be distinguished by the anchors. (See Figure 1(b) for an example.)



(a) Two Anchors Solve the Naming Problem.

(b) $(\sqrt{n}-1)^2+1$ anchors do not solve the naming problem.

Fig. 1. *min* and *any* for the grid

Lemma 10 (any). *If the anchors are chosen arbitrarily, then at least* $(\sqrt{n}-1)^2+2$ *anchors are needed to solve the naming problem.*

Proof. We prove the Lemma by giving an example where $(\sqrt{n}-1)^2+1$ anchors are already chosen, but there are still two nodes which are not distinguishable. Hence, at least $(\sqrt{n}-1)^2+2$ nodes are needed to solve the naming problem. The example is depicted in Figure 1(b). The white nodes are the anchors which are already chosen, whereas u, v are the nodes with same coordinate.

5.2 Routing

Given that the anchors a, b are placed as proposed in Lemma 8, a node u can compute the position of each anchor, based on its coordinate and the coordinate of its neighbors. It thus knows where the destination lies and hence can pass the message to one of its neighbors which lies in the quadrant of the destination guaranteeing that the message always reaches the destination on a shortest path.

Theorem 11. *The pseudo-geometric routing problem on the grid can be solved with two anchors. Furthermore, the chosen route between source and destination is a shortest path.*

6 Tree

6.1 Naming

Before we prove the minimal number of anchors needed to solve the naming problem in a tree, we define the following helpful term. Given a tree $T=(V,E)$, let the root r of the tree be an arbitrary node and call the such rooted tree T_r. Consider those nodes in T_r which have degree at least three and have no descendant with degree at least three. Formally $L(T_r)=\{v\in V; deg(v)\geq 3, \max_{u\in T_r(v)} deg(u)\leq 2\}$, where $deg(v)$ is the degree of a node v and $T_r(v)$ is the subtree of T_r rooted in v. Then the *minimal coverage number* $mc(T)$ of T is defined as $mc(T)=\max_{r\in V}\sum_{v\in L(T_r)}(deg(v)-2)$.

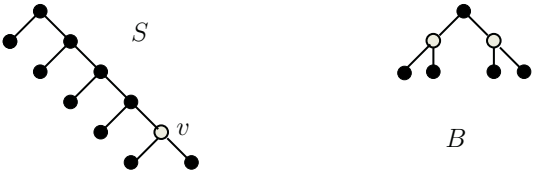


Fig. 2. Minimal coverage number in stair-tree S and complete binary tree B

To get an intuitive understanding of the minimal coverage number, we depict two examples in Figure 2.

In the stair-tree S, $L(S)=\{v\}$ and $mc(S)=deg(v)-2=1$. Note, that choosing v and the root as anchors also solves the naming problem.

In the complete binary tree $B = (V, E)$, $L(B) = \{v \in V; \exists l \in V, deg(l) = 1, d(v, l) = 1\}$, that is all nodes which are neighbors of a leaf. Then, $mc(B) = |L(B)| \cdot (3 - 2) = (n + 1)/4$. Again, note that choosing every second leaf as an anchor also solves the naming problem.

Lemma 12 (min). *Let $mc(T)$ be the minimal coverage number in a tree $T = (V, E)$. Then, we need at least $mc(T)$ anchors.*

Proof. Each node $u \in V$ with degree at least 3 must have at least one anchor in each but one of its neighbor-subtrees, where we use the term neighbor-subtree for subtrees of T rooted in neighbor nodes of u. Otherwise, if there are no anchors in more than one of its neighbor-subtrees, there are two neighbors u_1, u_2 of u which cannot be distinguished, since the distance from each anchor to u_1, u_2 is exactly the distance to u plus one for both. Hence, u_1 and u_2 have the same coordinate. Thus, the number of anchors we need is at least the minimal coverage number. (We have to subtract 2 from the degree of each node to avoid double counting and have a true lower bound.)

It is worthwhile to observe that the above discussion of Figure 2 shows that the minimal coverage number is –at least for some trees– (almost) tight.

By the Lemma above and the depicted example (Figure 2) the following lemma is self-explaining.

Lemma 13 (local). *It is not possible to solve the naming problem in a tree locally.*

Lemma 14 (any). *If the anchors are chosen arbitrarily, we need up to $n - 1$ nodes to solve the naming problem.*

Proof. If in the star we choose the center as an anchor, we additionally have to choose $n - 2$ of the $n - 1$ siblings in order to distinguish each pair of siblings.

Lemma 15. *In a tree it is always sufficient to choose all leaves as anchors.*

Proof. Consider two arbitrary nodes u, v in the tree. The nodes lie on a path p connecting u, v via their least common ancestor and furthermore connecting u, v to a leaf-anchor l. A leaf-anchor is a node with degree one on this path p. Hence, by Lemma 1, l can distinguish between all nodes on this path, specifically between u and v. This shows that any two nodes can be distinguished.

6.2 Routing

Assume now that each leaf is an anchor[2]. Based on its own coordinate and the coordinate of its neighbors a node u knows for each anchor in which direction it lies. By the choice of the anchors, the coordinate of the destination t is smaller than u's coordinate in at least one position i. Then t must lie in the direction of the corresponding anchor a_i, since in all other directions the distance to a_i is increasing. Thus, u passes the message to its neighbors which lies in this direction and consequently the message always reaches the destination on the shortest (and only) path.

[2] There are trees, where this choice is near to optimal, like the complete binary tree, but there are others, like the stair-tree in Figure 2, where choosing all leaves is wasteful.

Theorem 16. *The pseudo-geometric routing problem on the tree can be solved with $|L|$ anchors, where $|L|$ is the number of leaves in the tree.*

7 Unit Disk Graph

7.1 Naming

In order to prove a lower bound on the number of anchors needed in the unit disk graph[3] to solve the naming problem, we construct a unit disk graph which experiences this lower bound.[4] Specifically, we construct a unit disk grid tree, that is a unit disk tree, which is a subgraph of the grid graph. As we have seen in Section 6, trees with a large number of leaf-siblings (that is leaves which have a common father) have a large minimum coverage number and thus experience a large lower bound. Hence, we build a unit disk tree in such a way that the number of leaf-siblings is maximal. We henceforth show how the graph is constructed, lower bound the number of nodes in graph distance k from the root and then lower bound the total number of nodes in the whole graph. Based on those bounds we finally prove that there are $\Theta(n)$ sibling-leaves and hence, by Lemma 12 we need $\Theta(n)$ anchors to solve the naming problem.

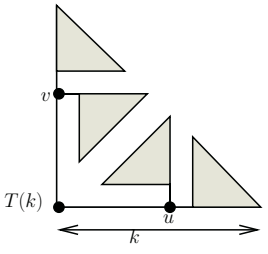


Fig. 3. Recursive construction of unit disk graph

The unit disk grid tree $T(k)$ is built recursively as depicted in Figure 3. The tree with depth k consists of four trees with depth $(k-3)/2$ each, depicted by shaded triangles in the figure. The root of the new tree is connected through two paths of length $(k+1)/2$ to two nodes u, v which are each connected themselves to two of the smaller trees.

Lemma 17. *In $T(k)$ there are at least $\frac{(k+3)^2}{8}$ nodes which are at distance k from the root and $\frac{(k+3)^2}{16}$ which are at distance $k-1$ from the root.*

[3] A *unit disk graph* is a graph where there is an edge between two nodes iff their Euclidean distance is at most one.

[4] The complete graph K_n is also a unit disk graph and experiences a lower bound of $n-2$, but from a practical point of view this graph is not interesting since each node can hear each other node and so a message can just be transmitted with maximal radio strength and is immediately received by the destination.Thus, the naming problem is not an issue at all.

Proof. Let $L(k)$ be the number of nodes in distance k from the root. Then

$$L(k) = 4L((k-3)/2)$$

and $L(1) = 2$. We now develop this equation recursively, resulting in

$$\begin{aligned} L(k) &= 4^i L(\frac{k - \sum_{j=0}^{i-1} 3 \cdot 2^j}{2^i}) = 4^{\log((k+3)/4)} L(1) \\ &= ((k+3)/4)^2 \cdot 2 = (k+3)^2/8. \end{aligned}$$

Let $L'(k)$ be the number of nodes in distance $k-1$. This number can be computed as above, with the exception that $L'(1) = 1$. Hence we immediately get

$$L'(k) = ((k+3)/4)^2 \cdot 1.$$

Lemma 18. *In $T(k)$ there are at most $9 \cdot k^2$ nodes.*

Proof. Let $N(k)$ be the number of nodes in the tree $T(k)$. Then

$$N(k) < 4N((k-3)/2) + 2k$$

and $N(1) = 3$. As before we develop this equation recursively, and get

$$\begin{aligned} N(k) &< 4^i N(\frac{k - \sum_{j=0}^{i-1} 3 \cdot 2^j}{2^i}) + \sum_{j=0}^{i-1} 2 \cdot 4^j \frac{k - \sum_{p=0}^{j-1} 3 \cdot 2^p}{2^j} \\ &= 4^t N(1) + 2(k(2^t - 1) - (4^t - 1) + 3(2^t - 1)) \leq 9 \cdot k^2, \end{aligned}$$

where we substituted t for $\log((k+3)/4)$ for the sake of readability.

Lemma 19 (min). *There are unit disk graphs where we need at least $((\sqrt{n}+9)/12)^2$ anchors to solve the naming problem.*

Proof. By Lemma 17 and the definition of the minimal coverage number in Section 6 the minimal coverage number of the constructed unit disk graph of depth k is $(k+3)^2/16$. Since by Lemma 18 the graph of depth k has at most $9k^2$ nodes, we can construct a unit disk graph on n nodes with depth at least $\sqrt{n}/3$. Substituting $\sqrt{n}/3$ for k we thus obtain a minimal coverage number of at least $((\sqrt{n}+9)/12)^2$. Following Lemma 12 this concludes the proof.

The locality-lemma is a direct implication of the lemma above.

Lemma 20 (local). *It is not possible to solve the naming problem in a unit disk tree locally.*

Lemma 21 (any). *If chosen arbitrarily, we need up to $n-1$ anchors to solve the naming problem in unit disk graphs.*

Proof. If we choose all nodes in $T(\sqrt{n}/3)$ except two sibling-leaves, we still cannot distinguish between those two siblings. Hence, we have to choose one of them to solve the naming problem and have in total $n-1$ anchors.

7.2 Routing

By the same discussion as in Section 6 the routing problem on the unit disk grid tree can be solved by choosing all leaves as anchors.

Theorem 22. *The pseudo-geometric routing problem on the unit disk grid tree can be solved with* $2 \cdot ((\sqrt{n}+9)/12)^2$ *anchors. Furthermore, the chosen route between source and destination is the shortest path.*

8 Butterfly

Due to the lack of space we omit the proofs for the lemmas in this section and refer the interested reader to the full paper [28].

8.1 Naming

We say that a node in the butterfly network is in column i if it is in the $(i+1)st$ column from the left (that is the leftmost column has index 0) and in row j if it is in the jth row from the top (that is the uppermost row has index 1) (see Figure 4). In a butterfly network on n nodes we have k rows and $(\log k + 1)$ columns, where $k(\log k + 1) = n$, that is $k = cn/\log n$, $c \leq 2$. An optimal choice of anchors, as shown in Lemma 23, is then to select all nodes in column 0 which are in row at most $k/2$ and all nodes in column $\log k$ which are in odd rows (see Figure 4, where the white nodes are anchors).

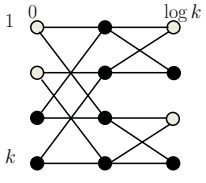


Fig. 4. k anchors solve the naming problem

Lemma 23 (min). *In the butterfly network we need* k *anchors to solve the naming problem, where* $k(\log k + 1) = n$.

Lemma 24 (local). *It is not possible to solve the naming problem in the butterfly locally.*

Lemma 25 (any). *If chosen arbitrarily, we need* $n - 1$ *anchors in the butterfly network to solve the naming problem.*

8.2 Routing

Given that the anchors are chosen as described above we obtain following result.

Theorem 26. *The pseudo-geometric routing problem on the butterfly network can be solved with k anchors, where $k(\log k + 1) = n$.*

9 Hypercube

9.1 Naming

If we subsequently use the term coordinate we mean the classical hypercube coordinate, with one bit per dimension. For example in a 2-dimensional hypercube the nodes have classical coordinates $(0,0), (0,1), (1,0)$ and $(1,1)$. If we refer to the (pseudo) coordinate as obtained by the anchors, we use the term distance vector. Furthermore, with d we refer to the dimension of the hypercube, where $d = \log n$.

Lemma 27 (min). *To solve the naming problem in a d-dimensional hypercube one needs at least $\log n / \log\log n$ anchors.*

Proof. Each anchor a is able to subdivide the nodes in at most d classes, since the distance of a node to a is at most d. Thus, k anchors are able to differentiate between at most d^k nodes. Since we need to differentiate between all nodes it must hold that $d^k \geq n$ and we immediately get $k \geq \log n / \log\log n$.

Lemma 28 (any). *If chosen arbitrarily, we need up to $n/4+1$ anchors in the hypercube to solve the naming problem*

Proof. The goal is to make the points $u = (0,0,0,\ldots,0)$ and $v = (1,1,0,\ldots,0)$ indistinguishable for as many anchors as possible. Towards this goal we choose all anchors to be of the form $(1,0,0,\ldots,0) + x$ and $(0,1,0,\ldots,0) + x$, where x is a d-dimensional vector with first and second coordinate zero, and the other entries can be chosen arbitrarily. The distance of each anchor to u and v is thus $1+|x|$, where $|x|$ is the number of ones in the coordinate of x. The number of anchors we have chosen, without being able to distinguish u, v is $2^{d-2} = n/4$. Hence, to solve the naming problem we need at least one further anchor and the lemma follows.

We subsequently show how to choose d anchors which solve the naming problem locally.

Lemma 29. *Suppose each node with distance one to the origin is an anchor. Then for each node u in the hypercube it holds that its distance vector is composed of at most two different values.*

Proof. The coordinate of a node p in the hypercube can be expressed in the coordinates of the anchors $a_1, \ldots, a_d$, where a_i has a 1 in position i of its coordinate, and zeroes elsewhere: $p = t_1 \cdot a_1 + t_2 \cdot a_2 + \ldots + t_d \cdot a_d$, where $t_i \in \{0,1\}$. Let Δ_i be the distance of p to a_i. Then, $\Delta_i = t_1 + \ldots + t_{i-1} + \neg t_i + t_{i+1} + \ldots t_d$, where $\neg 0 \equiv 1$ and $\neg 1 \equiv 0$. We claim that out of any three distances between one point p and three anchors, at least

two have to be equal. Without loss of generality (wlog) we consider the distances to the anchors a_1, a_2, a_3.

$$\begin{aligned}\Delta_1 &= \neg t_1 + t_2 + t_3 + t_4 + \ldots + t_d \\ \Delta_2 &= t_1 + \neg t_2 + t_3 + t_4 + \ldots + t_d \\ \Delta_3 &= t_1 + t_2 + \neg t_3 + t_4 + \ldots + t_d.\end{aligned}$$

Assume further wlog that $\Delta_1 \neq \Delta_2$. Thus, it has to hold that $t_1 \neq t_2$. We now show that Δ_3 has to be equal to either Δ_1 or Δ_2. First we assume that $\Delta_3 \neq \Delta_1$ and consequently $t_1 \neq t_3$. Then, $t_3 = t_2$ since $t_i \in \{0, 1\}$ and by assumption above $t_1 \neq t_2$. Therefore, $\Delta_2 = t_1 + \neg t_2 + t_3 + t_4 + \ldots + t_d = \Delta_3$. The same argument holds if $\Delta_3 \neq \Delta_2$ and in general for arbitrary three-tuples of distances and the lemma is proved.

Lemma 30 (local). *If each node with distance one to the origin is an anchor, all nodes in the hypercube can obtain unambiguously their coordinate as a function of their distance vector.*

Proof. The coordinate of node p in the hypercube can be expressed in the coordinates of the anchors $a_1, \ldots, a_d$: $p = t_1 \cdot a_1 + t_2 \cdot a_2 + \ldots + t_d \cdot a_d$, where $t_i \in \{0, 1\}$. By Lemma 30 each distance vector is comprised of at most two values x, y, where we assume wlog that $x < y$. We show how the coordinate of p is derived by its distance vector. Start with Δ_1 and Δ_2. There are two cases, either $\Delta_1 = \Delta_2$ or $\Delta_1 \neq \Delta_2$. In the first case, $\neg t_1 + t_2 + t_3 + \ldots + t_d = t_1 + \neg t_2 + t_3 + \ldots + t_d$, and hence $\neg t_1 + t_2 = t_1 + \neg t_2$, which means that $t_1 = t_2$. If on the other hand $\Delta_1 \neq \Delta_2$, we get $\neg t_1 + t_2 \neq t_1 + \neg t_2$. We can distinguish further whether $\Delta_1 < \Delta_2$ or not and get $\neg t_1 + t_2 < t_1 + \neg t_2$. if $\Delta_1 < \Delta_2$ or $\neg t_1 + t_2 > t_1 + \neg t_2$. if $\Delta_1 > \Delta_2$. In the first case the only feasible solution is $t_1 = 1, t_2 = 0$ in the second one $t_1 = 0, t_2 = 1$. The equations above hold for each pair of distances and hence we can deduce that if the distance vector is comprised of *exactly two different* values, then mapping the smaller value to one and the larger value to zero gives the exact coordinate of the node. The mapping is unambiguous, since there is only one feasible solution to the inequalities. If on the other hand the distance vector is comprised of *exactly one* value, we merely obtain the relation $t_1 = t_2 = \ldots = t_d$. Therefore, we basically have two possibilities of mapping the nodes: Either we map to the origin or to the coordinate $(1, 1, \ldots, 1)$. Since the distance vector of the origin contains only 1s, whereas the distance vector of the point $(1, 1, \ldots, 1)$ contains only ds, we can additionally distinguish between those two points and map $(1, 1, \ldots, 1)$ to $(0, 0, \ldots, 0)$ and $(d, d, \ldots, d)$ to $(1, 1, \ldots, 1)$ and the lemma is proved.

9.2 Routing

Assume now that each node with distance one to the origin is an anchor as proposed in Lemma 30. Based on its own, its neighbors' and the destination's distance vector each node u can compute its own, its neighbors' and the destination's coordinate. If u passes the message to the neighbor v with the smallest Hamming distance to the destination's coordinate, the message always reaches the destination on a shortest path.

Theorem 31. *The pseudo-geometric routing problem on the hypercube can be solved (locally) with* $\log n$ *anchors. Furthermore, the chosen route between source and destination is a shortest path.*

10 Conclusions

In this paper we proposed a new routing algorithm called pseudo-geometric routing, and analyzed it for the usual suspects of network topologies. We believe that pseudo-geometric routing may evolve into a promising new routing paradigm, for highly dynamic real world networks with memory constraints.

In the table below we summarize the results of the previous sections. For each topology we give the lower and upper bound, indicate by a checkmark whether a local solution exists and whether the routing scheme finds the shortest path between an arbitrary source and destination, or only an approximation.

	min	any	local	s.p.
Line	1	2	✓	✓
Ring	2	3	✓	✓
Grid	2	$(\sqrt{n}-1)^2+2$	–	✓
Tree	$mc(T)$	$n-1$	–	✓
UDG	$((\sqrt{n}+9)/12)^2$	$n-1$	–	✓
Butterfly	$\frac{n}{\log n}$	$n-1$	–	–
Hypercube	$\frac{\log n}{\log\log n}$	$n/4+1$	✓	✓

There are still a lot of questions open with regard to pseudo-geometric routing, or routing for dynamic networks in general. Most importantly, we plan to study the "real-world" behavior of our algorithm. In particular, the lower bound for unit disk graphs (which are a generally accepted model for all sorts of multihop radio networks) is rather discouraging. On the other hand, the result for highly regular unit disk graphs such as the grid, is encouraging. Real sensor networks will be somewhere in-between general unit disk graphs and a grid. In this context, an in-depth analysis of mobility vs. updates is also a direction of future research.

References

[1] I. Abraham, D.Dolev, and D. Malkhi. LLS: a Locality Aware Location Service for Mobile Ad hoc Networks. *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, 2004.

[2] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact Name Independent Routing with Minimum Stretch. *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2004.

[3] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with Guaranteed Delivery in Ad hoc Wireless Networks. In *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, 1999.

[4] C.Busch, S. Surapaneni, and S. Tirthapura. Analysis of Link Reversal Routing Algorithms for Mobile Ad hoc Networks. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 210–219. ACM Press, 2003.

[5] L. Cowen. Compact Routing with Minimum Stretch. In *Proc. of the ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 1999.

[6] L. Doherty, L. E. Ghaoui, and K. Pister. Convex Position Estimation in Wireless Sensor Networks. In *Proc. of Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, 2001.

[7] T. Eilam, S. Moran, and S. Zaks. A Simple DFS-Based Algorithm for Linear Interval Routing. In *WDAG '97: Proceedings of the 11th International Workshop on Distributed Algorithms*, 1997.

[8] K. Fath, P. Flocchini, and S. Pierre. A Compact Routing Technique for Communication Networks . *IEEE Canadian Conference on Electrical and Computer Engineering*, 1999.

[9] P. Fraigniaud and C. Gavoille. Interval Routing Schemes. *Algorithmica*, 21(2):155–182, 1998.

[10] P. Fraigniaud, C. Gavoille, and B. Mans. Interval Routing Schemes Allow Broadcasting with Linear Message-complexity. *Proc. of Symp. on Principles of Distributed Computing (PODC)*.

[11] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for Generating Loop-free Routes in Networks with Frequently Changing Topology. *IEEE Transactions on Communications*, 29:11–18, 1981.

[12] C. Gavoille. A Survey on Interval Routing. *Theoretical Computer Science*, 245(2):217–253, 2000.

[13] C. Gavoille. Routing in Distributed Networks: Overview and Open Problems. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 32, 2001.

[14] C. Gavoille and M. Gengler. Space-Efficiency for Routing Schemes of Stretch Factor Three. *Journal of Parallel and Distributed Computing*, 61(5):679–687, 2001.

[15] C. Gavoille and D. Peleg. Compact and Localized Distributed Data Structures. *Journal of Distributed Computing*, 16:111–120.

[16] L. Kleinrock and F. Kamoun. Hierarchical Routing for Large Networks. *Computer Networks*, 1:155 – 174, 1975.

[17] E. Kranakis, H. Singh, and J. Urrutia. Compass Routing on Geometric Networks. *11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[18] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Unit Disk Graph Approximation. In *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, 2004.

[19] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *Proc. of Symp. on Principles of Distributed Computing (PODC)*, 2003.

[20] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual Coordinates for Ad hoc and Sensor Networks. *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, 2004.

[21] D. Peleg and E. Upfal. A Tradeoff between Space and Efficiency forRouting Tables. In *Proc. of ACM Symp. on Theory of Computing (STOC)*, 1988.

[22] A. Rao, C. Papadimitriou, S. Ratnasamy, S. Shenker, and I. Stoica. Geographic Routing without Location Information. In *Proc. of Mobile Computing and Networking (MobiCom)*, 2003.

[23] N. Santoro and R. Khatib. Labelling and Implicit Routing in Networks. *Comput. J.*, 28(1):5–8, 1985.

[24] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz. Localization from Mere Connectivity. In *Proc. of Intl. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2003.
[25] H. Takagi and L. Kleinrock. Optimal Transmission Ranges for Randomly dDistributed Packet Radio Terminals. *IEEE Transactions on Communications*, 32:246–257, 1984.
[26] M. Thorup and U.Zwick. Compact Routing Schemes. *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2001.
[27] J. van Leeuwen and R. Tan. Interval Routing. *The Computer Journal*, 30:298 – 307, 1987.
[28] M. Wattenhofer, R. Wattenhofer, and P. Widmayer. Geometric routing without geometry. Technical report, ETH Zurich, 2005.

Author Index

Auletta, Vincenzo 3

Beimel, Amos 18
Bermond, Jean-Claude 34
Bilò, Davide 49
Braud, Laurent 34

Calamoneri, Tiziana 65
Caragiannis, Ioannis 78
Clementi, Andrea E.F. 89
Coudert, David 34

Das, Shantanu 99
De Prisco, Roberto 3
Di Ianni, Miriam 89
Dinitz, Yefim 115
Dobrev, Stefan 127

Fishkin, Aleksei V. 78
Flocchini, Paola 99
Fraigniaud, Pierre 140

Hinkelmann, Markus 155

Ilcinkas, David 140
Izumi, Taisuke 170

Jakoby, Andreas 155
Jansson, Jesper 127

Kaklamanis, Christos 78
Katreniak, Branislav 185
Klasing, Ralf 200
Královič, Rastislav 216
Královič, Richard 216

Lau, Francis C.M. 292
Lauria, Massimo 89

Markou, Euripides 200
Masuzawa, Toshimitsu 170
Monti, Angelo 89

Nayak, Amiya 99

Papaioannou, Evi 78
Penna, Paolo 3, 231
Persiano, Giuseppe 3
Prencipe, Giuseppe 246
Proietti, Guido 49

Radzik, Tomasz 200
Rajsbaum, Sergio 140
Rossi, Gianlucca 89

Sadakane, Kunihiko 127
Santoro, Nicola 99, 262
Sarracco, Fabiano 200
Shalom, Mordechai 277
Silvestri, Riccardo 89
Solomon, Noam 115
Sung, Wing-Kin 127

Tixeuil, Sébastien 140

Ventre, Carmine 231
Vocca, Paola 65

Wang, Rui 292
Wattenhofer, Mirjam 307
Wattenhofer, Roger 307
Widmayer, Peter 262, 307

Zaks, Shmuel 277