Boaz Patt-Shamir
Tınaz Ekim (Eds.)

LNCS 6058

Structural Information and Communication Complexity

17th International Colloquium, SIROCCO 2010
Şirince, Turkey, June 2010
Proceedings

Springer

Lecture Notes in Computer Science 6058

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

Boaz Patt-Shamir Tınaz Ekim (Eds.)

Structural Information and Communication Complexity

17th International Colloquium, SIROCCO 2010
Şirince, Turkey, June 7-11, 2010
Proceedings

Springer

Volume Editors

Boaz Patt-Shamir
School of Electrical Engineering, Tel Aviv University
Tel Aviv, 69978, Israel
E-mail: boaz@eng.tau.ac.il

Tınaz Ekim
Department of Industrial Engineering, Boğaziçi University
34342, Bebek-Istanbul, Turkey
E-mail: tinaz.ekim@boun.edu.tr

Library of Congress Control Number: Applied for

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

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

ISSN 0302-9743
ISBN-10 3-642-13283-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-13283-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The Colloquium on Structure, Information, Communication, and Complexity (SIROCCO) is devoted to the study of communication and knowledge in multiprocessor systems from both the qualitative and quantitative viewpoints. Special emphasis is given to innovative approaches and fundamental understanding, in addition to efforts to optimize current designs.

SIROCCO has a tradition of interesting and productive scientific meetings in a relaxed and pleasant atmosphere, attracting leading researchers in a variety of fields in which communication and knowledge play a significant role. Consistent with this tradition, the 17th SIROCCO meeting was held in Turkey, in the picturesque Nesin Mathematics Village, Şirince, İzmir, during June 7–11 2010.

Thirty-seven papers were submitted to SIROCCO 2010. All papers underwent a thorough peer-review process, where each submission was reviewed by three to six reviewers. The reviews were the basis of the Program Committee deliberations, which resulted in selecting 19 contributions for presentation at the colloquium and publication in this volume.

The presentations in this volume also include the abstract of an invited talk on communication complexity, given by Eyal Kushilevitz.

We thank the authors of all the submitted papers, the Program Committee members, and the external reviewers. Without their dedication, we could not have prepared a program of such quality.

We would also like to thank the SIROCCO Steering Committee Chair, Pierre Fraigniaud, for his energy and leadership in making this conference happen.

Last but not least, we would like to thank the local arrangements people from Nesin Mathematics Village, notably Aslı Can Korkmaz and the many students who volunteered on the organization team, for their invaluable help.

Boaz Patt-Shamir
Tınaz Ekim

2010 Prize for Innovation in Distributed Computing

The Prize for Innovation in Distributed Computing for 2010 was awarded to Jean-Claude Bermond (CNRS, INRIA Sophia-Antipolis, and Université Nice Sophia Antipolis). The prize was given in recognition of Bermond's contribution to the study of the impact of structure of networks on the efficiency of parallel or distributed algorithms, as illustrated by several papers that have appeared in the proceedings of past SIROCCO meetings. These papers tackled a wide variety of problems including routing, broadcasting, gossip protocols, traffic grooming, fault-tolerant network design, monopolies, and other topics.

2010 Prize Committee

Pierre Fraigniaud	CNRS & Université Paris Diderot, France
Shay Kutten	Technion, Israel
Nicola Santoro	Carleton University, Canada
Alexander A. Shvartsman	University of Connecticut, USA
Shmuel Zaks	Technion, Israel

Conference Organization

Program Committee

Amotz Bar-Noy	City University of New York, USA
Joffroy Beauquier	Université Paris-Sud 11 & LRI, France
Andrea Clementi	Università degli Studi di Roma "Tor Vergata", Italy
Tınaz Ekim	Boğaziçi University, Turkey
Robert Elsässer	Universität Paderborn, Germany
Pinar Heggernes	Universitetet i Bergen, Norway
Alex Kesselman	Google Inc., USA
Elias Koutsoupias	Univerity of Athens, Greece
Dariusz Kowalski	University of Liverpool, UK
Thomas Moscibroda	Microsoft Research, USA
Boaz Patt-Shamir (Chair)	Tel Aviv University, Israel
Thomas Sauerwald	Simon Fraser University & PIMS, Canada
Maria Serna	Universitat Politecnica de Catalunya, Spain
Peter Widmayer	Eidgenössische Technische Hochschule Zürich, Switzerland

Steering Committee

Pascal Felber	Université de Neuchâtel, Switzerland
Paola Flocchini	University of Ottawa, Canada
Pierre Fraigniaud (Chair)	CNRS & Université Paris Diderot, France
Lefteris Kirousis	University of Patras, Greece
Rastislav Kralovic	Comenius University, Slovakia
Evangelos Kranakis	Carleton University, Canada
Danny Krizanc	Wesleyan University, USA
Shay Kutten	Technion, Israel
Bernard Mans	Macquarie University, Australia
David Peleg	Weizmann Institute, Israel
Giuseppe Prencipe	Università di Pisa, Italy
Nicola Santoro	Carleton University, Canada
Alex Shvartsman	University of Connecticut, USA
Pavlos Spirakis	CTI & University of Patras, Greece
Shmuel Zaks	Technion, Israel
Janez Žerovnik	University of Ljubljana, Slovenia

Local Organization

Tınaz Ekim (Chair)	Boğaziçi University, Turkey
Arman Boyacı	Boğaziçi University, Turkey

External Reviewers

Yuichi Asahiro
Davide Bilò
Turker Biyikoglu
Maria Blesa
Arman Boyacı
André Brinkmann
Hajo Broersma
Tiziana Calamoneri
Jurek Czyzowicz
Shantanu Das
Bastian Degener
Miriam Di Ianni
Yann Disser
Stefan Dobrev
Frederic Dorn
Michael Elkin
Cesim Erten
Tomas Feder
Irene Finocchi
Tobias Friedrich
Takuro Fukunaga
Martin Gairing
Seth Gilbert
Luciano Gualà
Emanuele Guido Fusco
Dag Haugland
David Ilcinkas
Taisuke Izumi
Matt Johnson
George Karakostas
Ralf Klasing
Krzysztof Krzywdzinski
Michael Lampis
Johannes Langguth
Sophie Laplante
Fredrik Manne
Annalisa Massini
Brendan McKay
Daniel Meister
Stephane Messika
Lars Nagel
Nicolas Nisse
Adrian Ogierman
Can Ozturan
Aris Pagourtzis
Mostofa Patwary
Paolo Penna
Christophe Picouleau
Giuseppe Prencipe
Tomasz Radzik
Peter Robinson
Mariusz Rokicki
Laurent Rosaz
Kamil Sarac
Sanem Sariel Talay
Florian Schoppmann
Alper Sen
Riccardo Silvestri
Arun Somani
Mudhakar Srivatsa
Grzegorz Stachowiak
Dirk Sudholt
Christopher Thraves
Qin Xin
Rico Zenklusen
Huaming Zhang
Lisa Zhang
Michele Zito

Table of Contents

Communication Complexity: From Two-Party to Multiparty

Eyal Kushilevitz*

Abstract. We consider the multiparty communication complexity model, where k players holding inputs $x_1, \ldots, x_k$ communicate to compute the value $f(x_1, \ldots, x_k)$ of a function f known to all of them.

Yao's classic two-party communication complexity model [3] is the special case $k = 2$ (see also [2]). In the first part of the talk, we survey some basic results regarding the two-party model, emphasizing methods for proving lower-bounds.

In the second part of the talk, we consider the case where there are at least three parties ($k \geq 3$). The main lower bound technique for the communication complexity of such multiparty problems is that of *partition arguments*: partition the k players into two disjoint sets of players and find a lower bound for the induced *two-party* communication complexity problem. We discuss the power of partition arguments for both deterministic and randomized protocols. (This part is based on a joint work with Jan Draisma and Enav Weinreb [1].)

References

[1] Draisma, J., Kushilevitz, E., Weinreb, E.: Partition Arguments in Multiparty Communication Complexity. In: ICALP, pp. 390–402 (2009)

[2] Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)

[3] Yao, A.C.: Some complexity questions related to distributed computing. In: STOC, pp. 209–213 (1979)

* Computer Science Department, Technion – Israel Institute of Technology, Haifa. eyalk@cs.technion.ac.il. Research supported by grant 1310/06 from the Israel Science Foundation (ISF).

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, p. 1, 2010.

On the Impact of Local Taxes in a Set Cover Game*

Bruno Escoffier, Laurent Gourvès, and Jérôme Monnot

LAMSADE, CNRS FRE 3234, Université de Paris-Dauphine, 75775 Paris, France
{escoffier,laurent.gourves,monnot}@lamsade.dauphine.fr

Abstract. Given a collection $\mathcal{C}$ of weighted subsets of a ground set $\mathcal{E}$, the SET COVER problem is to find a minimum weight subset of $\mathcal{C}$ which covers all elements of $\mathcal{E}$. We study a strategic game defined upon this classical optimization problem. Every element of $\mathcal{E}$ is a player which chooses one set of $\mathcal{C}$ where it appears. Following a public tax function, every player is charged a fraction of the weight of the set that it has selected. Our motivation is to design a tax function having the following features: it can be implemented in a distributed manner, existence of an equilibrium is guaranteed and the social cost for these equilibria is minimized.

1 Introduction

We study a strategic game where each player should choose a facility from a set of facilities available to him. Each facility has a cost and each player must pay a tax for the facility that he has selected. This tax is a fraction of the facility's cost, and it decreases when the number of players who select the facility increases. This game is a model for many applications: facilities are services and every player is a client who selects the cheapest service.

The social cost is defined as the total cost of the facilities selected by at least one player, no matter how much the players pay. As a motivation, the cost of a facility can be an environmental cost and taxes can be a right to pollute. Then the environmental impact of the players' choice is much more important than the amount of money they pay.

In the game, no central authority can control the player's choice and minimize the social cost. Instead the players are self-interested, they all choose the facility that induces the lowest tax. We deal with the case where taxes are locally defined on the facilities (independently on the players' choices for other facilities) so it can be implemented in a completely distributed manner. Local tax functions induce a game among the players. The question posed in this article is *"Which local tax function minimizes the social cost?"*. Assuming that the game's outcome is an equilibrium, we study the *price of anarchy* (PoA in short) which is the worst case ratio between the social cost of a equilibrium, and the optimal social

* This work is supported by French National Agency (ANR), project COCA ANR-09-JCJC-0066-01.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 2–13, 2010.

cost [1]. In particular we consider pure strategy Nash equilibria and its robust refinements to strong equilibria and $k-$strong equilibria.

We first study a tax function where the cost of a facility is evenly shared by the players who select it. It is fair (the users of the a facility are treated equally) and budget balanced (the taxes cover the total cost). Next we investigate local tax functions which are fair, non-negative and monotone non-increasing in the number of players who select it. Any local tax function of this kind encourages the sharing of the facilities and it guarantees the existence of a pure strategy strong equilibrium.

2 Definitions, Related Work and Summary of Results

Set Cover. The situation described in introduction is usually modeled as a SET COVER problem. Given a set $\mathcal{E} = \{e_1, \ldots, e_n\}$ of n elements, a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m subsets of $\mathcal{E}$ such that $\bigcup_{j=1}^{m} S_j = \mathcal{E}$ and a weight function $w : \mathcal{S} \to \mathbb{R}_+$, the problem is to find $X \subseteq \mathcal{S}$ such that every element in $\mathcal{E}$ belongs to at least one member of X and $\sum_{S_j \in X} w(S_j)$ is minimum. In the following we sometimes write w_j (resp. s_j) to denote $w(S_j)$ (resp. $|S_j|$).

We study a SET COVER game defined upon the SET COVER problem. A facility j is associated with each set $S_j \in \mathcal{S}$. Each element $e_i \in \mathcal{E}$ is controlled by a player i who wants e_i to be covered by a set of $\mathcal{S}$. The set of facilities $\{1, \ldots, m\}$ and the set of players $\{1, \ldots, n\}$ are respectively denoted by M and N. Each player $i \in N$ has a strategy set Σ_i defined as $\{j \in M : e_i \in S_j\}$. We denote by $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$ the set of all states (or strategy profiles). The i-th coordinate of $a \in \Sigma$, denoted by a_i, is the action of i (actions are singletons). The congestion (or load) of a facility j is the number of players who want their element to be covered by S_j. It is denoted by $\ell_j(a)$ and defined as $|\{i \in N : a_i = j\}|$.

In the general *tailored model*, there is one function C which depends on locally available values: s_j, w_j and $\ell_j(a)$ for a given state a. This function C is unique, public and used locally by every facility. $C(s_j, w_j, \ell_j(a))$ is the tax that every player who has selected j must pay. We assume that C exhibits economies of scale, i.e. C is a monotone non increasing function of $\ell_j(a)$. In addition, C is non negative (players are not paid to select a facility). The goal in this model is to find a function that minimizes the social cost. If the taxes paid by the agents do not cover the cost of all selected facilities then we interpret it as the introduction of subsidies.

In the *Fair balanced model*, the cost of a facility is evenly shared by the players that chose this facility: $C(s_j, w_j, \ell_j) = w_j/\ell_j$. This function is a natural particular case of the tailored model. It corresponds to situations where we want a fair and budget balanced cost (the value of a state equals the sum payments of the agents, while this property does not necessary holds for other tax functions in the tailored model).

Finally, the tax that player i must pay under the state a is $C(s_{a_i}, w_{a_i}, \ell_{a_i}(a))$ but we often write $c_i(a)$ instead.

Solutions, existence and ratios. Much of the research in computational game theory has focused on the *Nash equilibrium* (NE in short). It is a state in which no single player can deviate and benefit. A *strong equilibrium* (SE in short) is a state in which no coalition of players can deviate and benefit [2]. This refinement of the NE is then more robust. Given $a \in \Sigma$, $i' \in N$ and $j' \in \Sigma_{i'}$, $(a \mid_{i'} j')$ denotes the strategy profile where every player $i \in N \setminus \{i'\}$ plays a_i whereas i' plays j'. A *pure* strategy Nash equilibrium is a state $a \in \Sigma$ where $\forall i \in N$, $\forall j \in \Sigma_i$, $c_i(a) \leq c_i(a \mid_i j)$. Pure means that every player chooses a strategy deterministically and since we only consider pure strategies, we often omit the adjective 'pure'.

Given $N' \subseteq N$ and $a, b \in \Sigma$, $(a \mid_{N'} b)$ is the strategy profile where i plays a_i (resp. i plays b_i) if $i \in N \setminus N'$ (resp. if $i' \in N'$). A strong equilibrium is a state $a \in \Sigma$ where there is no coalition $N' \subseteq N$ and $b \in \Sigma$ such that $\forall i \in N'$, $c_i(a \mid_{N'} b) < c_i(a)$. A k-strong equilibrium is defined similarly for coalitions involving at most k players.

The set of Nash equilibria (resp. strong equilibria, k-strong equilibria) of a game Γ (the SET COVER game in our case) is denoted by $NE(\Gamma)$ (resp. $SE(\Gamma)$, $kSE(\Gamma)$). In particular $SE(\Gamma) \subseteq kSE(\Gamma) \subseteq NE(\Gamma) \subseteq \Sigma$.

A state is said optimal if it minimizes the social cost. The minimal social cost is denoted by OPT. We denote by $NASH$ (resp. $STRONG$, $kSTRONG$) the social cost of a given a Nash equilibrium (resp. strong equilibrium, k-strong equilibrium).

The price of anarchy (PoA) is a widely accepted performance measure of decentralized systems which relies on Nash equilibria [1]. It was generalized to $k-$strong equilibria and named the k-strong price of anarchy ($k-$SPoA in short) [3]. The k-SPoA is the worst case ratio between the weight of a pure k-strong equilibrium (assuming one exists) and the optimum, over all instances of a game. The PoA and the strong price of anarchy (SPoA in short) correspond to the cases $k = 1$ and $k = n$ respectively.

Related work. The SET COVER game is a *congestion game* where strategies are singletons. Rosenthal [4] showed that every congestion game possesses a pure strategy NE. The existence of SE in congestion games was first studied by Holzman and Law-Yone [5] who focused on situations where congestion has a negative effect on the players' utility (*monotone-decreasing* congestion games). After that, Rozenfeld and Tennenholtz [6] completed the study for *monotone-increasing* congestion games where congestion has a positive effect on the players' utility. In particular they prove that monotone-increasing congestion games where strategies are singletons always has a pure strategy strong equilibrium. In addition, this SE can be computed efficiently. The SET COVER game under the tailored model is a monotone-increasing congestion game so it always possesses a SE.

In [7], Anshelevich et al. study a connection game consisting of a network (an edge-weighted graph) and a pair (s_i, t_i) of nodes for each player i. A player's strategy is a set of edges that form an s_i-t_i path. In the unweighted case (all players have the same importance), the cost of each selected edge is shared equally by its users, i.e. if k agents use an edge of cost w in their strategy then

each of these agents pays a share of w/k. The SET COVER game under the fair balanced model is a particular case of Anshelevich et al.'s connection game if the network is as follows: we are given a supersource x, a node y_j and an edge (x, y_j) of cost w_j for each set S_j, a node z_i for each element e_i and an edge (y_j, z_i) of cost 0 iff $e_i \in S_j$. Each player i wants to connect x to z_i. The PoA of Anshelevich et al.'s connection game is n (the number of players) and the price of stability (the ratio of the best Nash equilibrium relative to the social optimum) is $H(n) = O(\log n)$ [7] where $H(n) = 1 + 1/2 + \cdots + 1/n$.

The connection game is also studied by Epstein et al. [8] who give topologic conditions for the existence of a strong equilibrium and bounds on the k-SPoA. In particular, when all players have the same source but not necessarily the same sink (single source), there is a strong equilibrium if the network is a series parallel graph [8]. This result applies for the "network" representation of the SET COVER game given above. For the k-strong price of anarchy of the connection game Epstein et al. prove that $\max\{\frac{n}{k}, H(n)\} \leq k\text{-SPoA} \leq \frac{n}{k}H(k)$. Therefore the PoA is n and the SPoA is $H(n)$.

In this paper we consider the fair balanced model like in [7,8] but we also investigate general local taxes (tailored model). The goal is to find one that induces the best system's efficiency. In other words, we aim at mitigating the system's deterioration due to selfish and uncoordinated behavior. In this line, Christodoulou et al. introduced the notion of *coordination mechanism* [9]. Recently coordination mechanisms received a lot of attention, in particular for scheduling games.

Finally the SET COVER game under consideration in this paper is close to the model studied by Buchbinder et al. [10]. A central authority encourages the purchase of resources by offering subsidies that reduce their price: every player who has selected resource j pays $(w_j - \sigma_j)/\ell_j$ where σ_j is the subsidy associated with j. Hence the fair balanced model corresponds to the case where $\sigma_j = 0$ for all j. The main differences with our model are that the players are introduced sequentially by an adversarial scheduler and the total amount of subsidies offered is bounded by the amount of non refundable taxes collected when a player purchases a new set.

Summary of results. We study in Section 3 the k-SPoA of the SET COVER game in the fair balanced model. We prove that $k\text{-SPoA} = H(k) - 1 + \frac{\Delta}{k}$ where Δ is the maximum size of a set. We deduce that $\text{PoA} = \Delta$ and $\text{SPoA} = H(\Delta)$. We also deduce that $k\text{-SPoA} = H(k) - 1 + \frac{n}{k}$, $\text{PoA} = n$ and $\text{SPoA} = H(n)$ where n is the number of players. These bounds are tight. When $k = 1$ and $k = n$, they meet the previous results given in [7,8]. However, when $1 < k < n$, we get a more accurate value of the k-SPoA, i.e. $H(k) - 1 + \frac{n}{k}$ instead of the $\frac{n}{k}H(k)$ given by Epstein et al. [8].

In Section 4, we study the tailored model. We first show that the PoA can reduce by more than 1/3 using a function that encourages players to choose big sets. On the other hand, we also show in this section strong lower bounds valid for *any* tax function. Bounds on the PoA obtained in the tailored model are of the same order as those obtained with the fair balanced model (see Table 1 for

Table 1. Upper and lower bounds on the k-SPoA as a function of (left part) n and (right part) Δ in the tailored model

	n			Δ		
	PoA	SPoA	k-SPoA	PoA	SPoA	k-SPoA[a]
UB	$0.66n$	$H(n)$	$H(k)-1+\frac{n}{k}$	Δ	$H(\Delta)$	$H(k)-1+\frac{\Delta}{k}$
LB	$0.5n$	$\frac{\ln(n)}{2}+O(1)$	$\max\left(\frac{\ln(n)}{2}+O(1), \frac{n}{2k}\right)$	Δ	$H(\Delta)$	$H(k)-1+\frac{\Delta}{k}$

[a] when $k \leq \Delta$. When $k > \Delta$, the k-SPoA is $H(\Delta)$.

a summary of the results given in this article). Then local taxes cannot (in the model dealt with here) considerably reduce the social cost (compared to a fair and balanced division of the cost).

Finally the case of uniform weights is studied. We mainly prove a lower bound of $n/4$ and an almost matching upper bound of $n/4 + 1/2 + 1/4n$ for the PoA. These bounds hold in the tailored model whereas PoA$= n$ in the fair balanced model.

Due to space limitations, some proofs are omitted.

3 Fair Balanced Model

We give in Theorem 1 the $k-$SPoA of the SET COVER game. Note that w.l.o.g., we can assume that $k \leq \Delta$ since any coalition N' can be decomposed into coalitions $N'_1, \ldots N'_p$ where each player of N'_i will play the same set S_{j_i} for $i = 1, \ldots, p$. The coalition N' strictly decreases the cost of each player in N' iff for every $i \in \{1, \ldots, p\}$, the coalition N'_i strictly decrease the cost of each player in N'_i. Now, since $|N'_i| \leq |S_{j_i}| \leq \Delta$, the result follows.

Theorem 1. *For any $k \in \{1, \ldots, \Delta\}$, the $k-$SPoA of the* SET COVER *game is $H(k) - 1 + \frac{\Delta}{k}$.*

Proof. Suppose that the players have reached a k-strong equilibrium a, i.e., there is no coalition $N' \subseteq N$ ($|N'| \leq k$) and $b \in \Sigma$ such that $c_i(a\,|_{N'}\, b) < c_i(a)$ for all $i \in N'$.

Given $j \in [1..m]$ and $x \in [1..|S_j|]$, $h(x, j)$ designates the player in S_j with the $x-$th largest cost. Hence $c_{h(1,j)} \geq c_{h(2,j)} \geq \ldots \geq c_{h(|S_j|,j)}$ holds. Ties are broken arbitrarily. Let us show that

$$c_{h(x,j)} \leq \frac{w_j}{\min\{x, k\}} \tag{1}$$

holds for all $j \in [1..m]$ and $x \in [1..|S_j|]$. By contradiction, assume that there are $j_0 \in [1..m]$ and $x_0 \in [1..|S_0|]$ such that $c_{h(x_0,j_0)} > \frac{w(S_{j_0})}{\min\{x_0,k\}}$. By assumption we get:

$$\forall x \in \{1, \ldots, \min\{x_0, k\}\}, \quad c_{h(x,j_0)} > \frac{w(S_{j_0})}{\min\{x_0, k\}} \tag{2}$$

since $c_{h(x,j_0)} \geq c_{h(\min\{x_0,k\},j_0)} \geq c_{h(x_0,j_0)}$. Now, let us consider the coalition $N' = \{h(x,j_0) : x = 1,\ldots,\min\{x_0,k\}\} \cap \{i \in N : a_i \neq j_0\}$ (where a is the k-strong equilibrium). Obviously, $|N'| \leq k$ and we get $N' \neq \emptyset$ since otherwise $\{h(x,j_0) : x = 1,\ldots,\min\{x_0,k\}\} \subseteq S_{j_0}$ and then, the cost of each player in $\{h(x,j_0) : x = 1,\ldots,\min\{x_0,k\}\}$ will be at most $\frac{w(S_{j_0})}{\min\{x_0,k\}}$, contradiction with inequality (2). If all the players in N' change their mind, form a coalition and simultaneously decide to be covered by S_{j_0} then their individual cost would be at most $\frac{w(S_{j_0})}{\min\{x_0,k\}}$. We deduce that the solution is not a k-strong equilibrium, contradiction.

Using (1) and the harmonic function $H(p) := \sum_{i=1}^{p} 1/i$, we deduce that

$$\begin{aligned} \sum_{i\in S_j} c_i &\leq \left(\sum_{i=1}^{\min\{|S_j|,k\}} \tfrac{1}{i} + \tfrac{|S_j|-\min\{|S_j|,k\}}{\min\{|S_j|,k\}}\right) w_j \\ &= \left(H(\min\{|S_j|,k\}) - 1 + \tfrac{|S_j|}{\min\{|S_j|,k\}}\right) w_j \\ &\leq \left(H(\min\{\Delta,k\}) - 1 + \tfrac{\Delta}{\min\{\Delta,k\}}\right) w_j \end{aligned} \tag{3}$$

holds for all $j \in \{1,\ldots,m\}$ since $|S_j| \leq \Delta$. Let $X \subseteq \mathcal{S}$ be an optimal solution to the underlying SET COVER problem with value $OPT = \sum_{S_j\in X} w_j$. Summing up inequalities (3) for the sets in X, we obtain:

$$\sum_{S_j\in X}\sum_{i\in S_j} c_i \leq \sum_{S_j\in X}\left(H(\min\{\Delta,k\}) - 1 + \frac{\Delta}{\min\{\Delta,k\}}\right) w_j$$

On the one hand $kSTRONG = \sum_{i\in N} c_i \leq \sum_{S_j\in X}\sum_{i\in S_j} c_i$ holds because X is feasible, i.e. each element $e \in \mathcal{E}$ is covered by at least one set in X. On the other hand, we get: $\sum_{S_j\in X}\left(H(\min\{\Delta,k\}) - 1 + \frac{\Delta}{\min\{\Delta,k\}}\right) w_j = \left(H(\min\{\Delta,k\}) - 1 + \frac{\Delta}{\min\{\Delta,k\}}\right) OPT$. A tight example (omitted for space reason) exists. □

From Theorem 1, we deduce that the PoA of the SET COVER game is Δ (set $k = 1$) and the SPoA of the SET COVER game is $H(\Delta)$ (set $k = \Delta$).

As a corollary, since $\Delta \leq n$, we get that the k-SPoA is at most $H(k) - 1 + \frac{n}{k}$, and hence that the PoA is at most n and the SPoA at most $H(n)$. These bounds are actually tight because $\Delta = n$ in the example showing the lower bound in the proof of Theorem 1 (omitted for space reason).

Corollary 1. *For any $k \in \{1,\ldots,n\}$, the k-SPoA of the* SET COVER *game is $H(k) - 1 + \frac{n}{k}$.*

4 Tailored Model: Finding the Right Taxes

The goal is to find a function C which minimizes the social cost. C is a non negative function monotone non increasing with $\ell_j(a)$. Equilibria are defined with respect to the taxes paid by players but the social cost remains the total weight of the selected sets. In contrast with the fair balanced model, we do not impose that the taxes cover the whole cost.

4.1 Improvement on the PoA

We use a function where the players are encouraged to select large sets (function decreasing in $s_j = |S_j|$) with a lot of other players (function decreasing in $\ell_j(a)$). We first show a lemma satisfied by these functions, and then study a particular class of functions that leads to interesting bounds (Lemma 2 and Theorem 2).

Fix a cost function and suppose that we have a NE strategy profile a (for this cost function). After a possible relabeling, we denote by $Y = \{S_1, S_2, \cdots, S_t\}$ the set of sets chosen by at least one player with $s_1 \geq s_2 \geq \cdots \geq s_t$. For the sake of clarity, suppose that $a_i = i$ for $i \leq t$ (i.e. player i chooses S_i).

Lemma 1. *Let c be a cost function which is decreasing in $\ell_j(a)$. Then, for any $i \in \{1, 2, \cdots, t\}$, $s_i \leq n + 1 - i$.*

Proof. Suppose that there exists i such that $s_i \geq n + 2 - i$. Since s_k are sorted in non increasing order, for any $k \leq i$ $s_k \geq n + 2 - i$. Then fix some $j \leq i$. Let $N' = \{1, \cdots, i\} \setminus \{j\}$. Since $s_j \geq n + 2 - i$, $s_j + |N'| \geq n + 1$ and consequently there exists a player $k \in N'$ with $e_k \in S_j$. Then, since the profile is a NE, $c_k(a) \leq c_k(a|_k j)$. Since c is decreasing in $\ell_j(a)$, $c_k(a) < c_j(a)$.

In the strategy profile a, for any $j \leq i$ there exists a $k \leq i$ such that $c_k(a) < c_j(a)$. This is obviously impossible. □

We now study a particular class of local tax functions $\frac{w_{a_i}}{g(s_{a_i}) + \epsilon g(\ell_{a_i}(a))}$ where g is a positive and increasing function and $\epsilon > 0$. Fix an optimal Set Cover $\{S_1^*, S_2^*, \cdots, S_{t^*}^*\}$, and let $s_j^* = |S_j^*|$.

Lemma 2. *Let the tax function be $\frac{w_{a_i}}{g(s_{a_i}) + \epsilon g(\ell_{a_i}(a))}$ where g is a positive and increasing function. Then:*

$$NASH \leq (1 + \epsilon) \sum_{j=1}^{t^*} \left(w(S_j^*) \frac{\sum_{i=1}^{s_j^*} g(n + 1 - i)}{g(s_j^*)} \right)$$

Proof. As previously, suppose that player i chooses S_i ($i \leq t$) and let $\lambda(j)$ (for $j \leq t^*$) be the set of players among $\{1, 2, \cdots, t\}$ which belong to S_j^* ($\lambda(j)$ is possibly empty).

Since g is increasing, the cost $c_i(a)$ associated to player i verifies:

$$c_i(a) = \frac{w(S_i)}{g(s_i) + \epsilon g(\ell_i(a))} \geq \frac{w(S_i)}{(1 + \epsilon) g(s_i)} \tag{4}$$

On the other hand, since the profile a is a Nash equilibrium (and since g is non negative), if e_i belongs to the set S_j^*:

$$c_i(a) \leq c_i(a|_i j) \leq \frac{w(S_j^*)}{g(s_j^*)} \tag{5}$$

From Equations (4) and (5) we get:

$$w(S_i) \leq (1+\epsilon) w(S_j^*) \frac{g(s_i)}{g(s_j^*)}$$

where j is such that $e_i \in S_j^*$. Summing up this inequality for all players $i \in \lambda(j)$, we get:

$$\forall j \in \{1, 2, \cdots, t^*\}, \sum_{i \in \lambda(j)} w(S_i) \leq (1+\epsilon) w(S_j^*) \frac{\sum_{i \in \lambda(j)} g(s_i)}{g(s_j^*)} \tag{6}$$

Since $s_i \leq n+1-i$ (thanks to Lemma 1), and since $g(s_i)$ is non negative and increasing, we get:

$$\sum_{i \in \lambda(j)} g(s_i) \leq \sum_{i=1}^{|\lambda(j)|} g(n+1-i) \leq \sum_{i=1}^{s_j^*} g(n+1-i) \tag{7}$$

Since the sets S_j^* cover all the elements, each player i belongs to at least one $\lambda(j)$. Then, summing Inequality (6) for $j \in \{1, 2, \cdots, t^*\}$, we get using (7):

$$\sum_{i=1}^{t} w(S_i) \leq \sum_{j=1}^{t^*} \sum_{i \in \lambda(j)} w(S_i) \leq (1+\epsilon) \sum_{j=1}^{t^*} \left(w(S_j^*) \frac{\sum_{i=1}^{s_j^*} g(n+1-i)}{g(s_j^*)} \right) \qquad \square$$

We are now able to find the PoA associated to the cost function under consideration.

Theorem 2. *Let the tax function be* $c_i(a) = \frac{w_{a_i}}{g(s_{a_i}) + \epsilon g(\ell_{a_i}(a))}$ *where* g *is a positive and increasing function. Then*

$$\frac{f(n)}{1+\epsilon} \leq PoA \leq (1+\epsilon) f(n)$$

where $f(n) = \max_{i=1,2\cdots,n} \left\{ \frac{\sum_{k=1}^{i} g(n+1-k)}{g(i)} \right\}$.

Proof. The upper bound is a straightforward consequence of Lemma 2.

For the lower bound, consider the following instance with n players and $p+2$ resources where $S_i = \{i, i+1, \cdots, n\}$ for $i = \{1, 2, \cdots, p+1\}$ and $S_0 = \{1, 2, \cdots, p\}$. We set $w_{p+1} = 0$ and fix the weights in such a way that the strategy where player $i \leq p$ chooses S_i (and player $i > p$ chooses S_{p+1}) is a NE. We fix for $i = 1, 2, \cdots, p$:

$$w(S_i) = g(s_i) + \epsilon g(1) = g(n+1-i) + \epsilon g(1)$$

which ensures that $\frac{w(S_i)}{g(s_i)+\epsilon g(2)} \geq \frac{w(S_{i+1})}{g(s_{i+1})+\epsilon g(1)}$ for $i = 1, 2, \cdots, p-1$. In particular, the cost associated to player i is 1 if $i \leq p$ and 0 if $i \geq p+1$. We set $w(S_0) = g(p) + \epsilon g(1)$, ensuring that players have no incentive to deviate.

Then we have a NE of value $\sum_{i=1}^{p}(g(n+1-i)+\epsilon g(1))$ while the solution consisting of taking only S_0 and S_{p+1} is a Set Cover of value $g(p)+\epsilon g(1)$. Then the PoA is such that:

$$PoA \geq \frac{p\epsilon g(1)+\sum_{i=1}^{p} g(n+1-i)}{\epsilon g(1)+g(p)} \geq \frac{\sum_{i=1}^{p} g(n+1-i)}{g(p)(1+\epsilon)}$$

where the last inequality holds since g is increasing. □

Note that the lower bound would hold for any cost function $\frac{w_j}{g(s_j)+h(\ell_j)}$ where g and h are increasing functions.

Now we derive from Theorem 2 a PoA smaller than n by choosing interesting functions g. Since g is non decreasing, we have:

$$\int_{k=n-i}^{n} g(x)dx \leq \sum_{k=1}^{i} g(n+1-k) \leq \int_{k=n+1-i}^{n+1} g(x)dx$$

If we choose $g(i)=i^{\alpha}$ for some $\alpha>0$, we get:

$$\frac{n^{\alpha+1}-(n-i)^{\alpha+1}}{\alpha+1} \leq \sum_{k=1}^{i} g(n+1-k) \leq \frac{(n+1)^{\alpha+1}-(n+1-i)^{\alpha+1}}{\alpha+1}$$

Studying this expression, if we choose $\alpha=1/2$, then we get a PoA equivalent to βn (up to a factor $1+\epsilon$) where $\beta=\frac{3-\sqrt{3}}{\sqrt{2}3^{1/4}}<0.69$. The best choice for α is actually $\alpha\simeq 0.643$ which gives a PoA equivalent to γn (up to a factor $1+\epsilon$) where $\gamma\simeq 0.660$.

4.2 Lower Bounds

We first deal with lower bounds which depend on Δ (and k for k-SPoA). As shown in Section 3, with the fair division of the cost we get a PoA of $H(k)-1+\frac{\Delta}{k}$ ($k\leq\Delta$). Here, we show that despite the quite important relaxation we consider (choosing the tax function), the social cost cannot be lowered down since no cost function can lead to a better k-SPoA.

Before giving lower bounds, we state a preliminary lemma (proof omitted).

Lemma 3. *Suppose that there exist $d\geq 1$, $k\geq s\geq 1$ and $w>0$ such that $C(d,0,d)>C(s,w,s)$. Then the $k-$SPoA of the* SET COVER *game is unbounded.*

It follows that every non-increasing tax function inducing a bounded $k-$SPoA satisfies $C(d,0,d)\leq C(s,w,\ell)$ where $1\leq\ell\leq s$ and $w>0$. Now we are ready to prove lower bounds.

Proposition 1. *Under the tailored model, and for any $k\in\{1,\cdots,\Delta\}$, the $k-$SPoA of the* SET COVER *game is at least $H(k)-1+\frac{\Delta}{k}$.*

Proof. Let $k \in \{1, \cdots, \Delta\}$. We consider an array of $(\Delta+1) \times (\Delta!)$ elements that we arrange in $\Delta+1$ lines and $\Delta!$ columns. For $q = 1, \cdots, \Delta!$, a set S_q contains the elements of column q from line 1 to line Δ. These sets are called *vertical*. For $l = 1, \cdots, k-1$, line l is partitioned into $(\Delta!)/l$ sets of size l. For $l = k, \cdots, \Delta$, line l is partitioned into $(\Delta!)/k$ sets of size k. Line $\Delta+1$ is partitioned into $(\Delta!)/\Delta$ sets of size Δ. These sets are called *horizontal*. Actually, we complete each such horizontal set by adding (at random) elements from line $\Delta+1$ in such a way that each horizontal set has size Δ. Hence, for sets from line $l < k$ we add $\Delta - l$ elements, and for sets from line l, $k \leq l \leq \Delta$, we add $\Delta - k$ elements. Vertical sets have weight one, and horizontal sets of line between 1 and Δ have weight one, horizontal sets of line $\Delta+1$ have weight 0.

Let us consider the state where every element of line $\Delta+1$ decides to be covered by the horizontal set where it appears (of weight 0), while in the other lines every element decides to be covered by the (unique) horizontal set where it appears. Hence the cost incurred by an element of line l is $C(\Delta, 1, l)$ for $l < k$, $C(\Delta, 1, k)$ for $k \leq l \leq \Delta$, and $C(\Delta, 0, \Delta)$ for $l = \Delta+1$. As a consequence of Lemma 3, we can assume that the elements of a set of weight 0 choose this set. If $r \leq k$ elements move to the same vertical set then at least one of them does not benefit since the new cost is $C(\Delta, 1, r)$ while the lowest previous cost, thanks to the fact that C is non increasing with $\ell_j(a)$, was at most $C(\Delta, 1, r)$. Then the state is a k-strong equilibrium. The optimum solution (made of all vertical sets plus the sets of weight 0) has weight $\Delta!$, while the k-strong equilibrium we found has weight $\left(\sum_{l=1}^{k-1} \frac{\Delta!}{l}\right) + \frac{\Delta!}{k}(\Delta+1-k) = \Delta!\left(H(k) - 1 + \frac{\Delta}{k}\right)$. □

For $k = \Delta$ (and more generally for $k \geq \Delta$) this gives $H(\Delta)$. For $k = 1$, it shows that the PoA is at least Δ.

Let us consider now lower bounds in terms of n (and k for $k-$SPoA). The instance given in Proposition 1 does not give a good lower bound since n is very big (the lower bound is $H(k)$ for $k \leq \Delta$ but $H(\Delta)$ for $k \geq \Delta$, which is not good since Δ is very small with respect to n). In the following proposition, we show an almost tight lower bound of $\max\left(\frac{\ln(n)}{2} + O(1), \left\lfloor \frac{n+k}{2k} \right\rfloor\right)$. Remember that the upper bound is $H(k) - 1 + \frac{n}{k}$, which is of order of $\ln(n)$ if $k \geq n/\ln(n)$ (as the lower bound) and of order of n/k if $k \leq n/\ln(n)$ (as the lower bound)[1].

For $k = 1$, the lower bound is $\left\lfloor \frac{n+1}{2} \right\rfloor \geq \frac{n}{2}$, which is quite close to the PoA of $0.66n$ obtained in Section 4.1.

Proposition 2. *Under the tailored model, and for any $k \in \{1, \cdots, \Delta\}$, the $k-$SPoA of the* SET COVER *game is at least* $\max\left\{\frac{\ln(n)}{2} + O(1), \left\lfloor \frac{n+k}{2k} \right\rfloor\right\}$.

Proof. Fix some n. To get the first lower bound, we adapt the instance given in Proposition 1 by trying to reduce the number of elements. Let t be the largest integer such that $t^2 \leq n$. We consider a square of t times t elements, and $n - t^2$ extra elements. We consider t vertical sets (the columns of the square) of weight 1, and in line i $\left\lfloor \frac{t}{i} \right\rfloor$ disjoint sets of size i. We add to these sets $t - i$ elements (at

[1] The largest ratio between these bounds is 4, when k is of order of $n/\ln(n)$.

random) from line t. We group all the elements not covered by these horizontal sets (including the $n-t^2$ extra ones) in one set S of weight 0. All other sets have weight 1. Note that all the sets (except possibly S) have size t.

We consider the solution where each player in set S chooses S, players in line t choose the set which contains this line, and players in line from 1 to $t-1$ choose the unique horizontal set to which they belong. By a similar reasoning, this is a strong equilibrium, of value $\sum_{i=1}^{t} \left\lfloor \frac{t}{i} \right\rfloor$. However $\sum_{i=1}^{t} \left\lfloor \frac{t}{i} \right\rfloor \geq \sum_{i=1}^{t} \left(\frac{t}{i} - 1\right) \geq t(H(t)-1)$. The solution consisting of set S plus the vertical sets has weight t, hence the ratio is at least $H(t) - 1 = \frac{\ln(n)}{2} + O(1)$ (since $\sqrt{n} \geq t \geq \sqrt{n} - 1$).

Now, we deal with the second lower bound. Fix n and k. Let $\lambda = \left\lfloor \frac{n+k}{2k} \right\rfloor$ and $p = \lambda k$. We consider the following instance, consisting of n players and $\lambda + 2$ sets: $S_0 = \{1, 2, \cdots, p\}$, $S_{\lambda+1} = \{p+1, p+2, \cdots, n\}$ and for $i = 1, 2, \cdots, \lambda$ $S_i = \{k(i-1)+1, k(i-1)+2, \cdots, ik\} \cup \{p+1, p+2, \cdots, 2p-k\}$. $S_{\lambda+1}$ has weight 0 while all other sets have weight 1. Note that this construction is possible since $2p - k \leq n$. Indeed, $p = \lambda k \leq \frac{n+k}{2}$.

Then the strategy profile where player $(i-1)k + j$ chooses S_i for $i \leq \lambda$ and $j \leq k$, and player i chooses $S_{\lambda+1}$ for $i \geq p+1$ is a k-strong equilibrium. Indeed since $S_{\lambda+1}$ has weight 0 players in $S_{\lambda+1}$ have no interest to change (as we said in Lemma 3), and a coalition of $r \leq k$ players (between 1 and p) changing their mind and choosing set S_0 would pay the same amount $C(p, 1, r)$ while their current cost is $C(p, 1, k) \leq C(p, 1, r)$ since the cost is non increasing with $\ell_j(a)$. The weight of this k-strong equilibrium is λ while the Set Cover consisting of taking S_0 and S_{p+1} has weight 1. □

4.3 Uniform Weights

In Section 4.2, the lower bounds are obtained with a set of weight 0. Then, one can wonder whether these lower bounds still hold when the weight of any set is fixed (for instance weight one). We show almost similar lower bounds for this case (Propositions 3, 4 and 5). As a final result, the PoA drops to $n/4$ in the uniform case (PoA$= 0.66n$ in the tailored model and PoA$= n$ in the fair balanced model).

Proposition 3. *Under the tailored model, and for any $k \in [1..\Delta]$, the $k-SPoA$ of the* SET COVER *game is at least $H(k) - 1 + \frac{\Delta}{k} - \frac{1}{k} + \frac{1}{\Delta}$ if weights are uniform.*

For $k = \Delta$ (and more generally for $k \geq \Delta$) this gives $H(\Delta)$. For $k = 1$, the bound is $\Delta - 1 + \frac{1}{\Delta}$ but the next Proposition gives a better bound of Δ.

Proposition 4. *Under the tailored model, the PoA of the* SET COVER *game is at least Δ if weights are uniform.*

Finally, dealing with n, we can also get strong lower bounds for uniform weights.

Proposition 5. *Under the tailored model, and for any $k \in [1..\Delta]$, the k-SPoA of the* SET COVER *game is at least $\frac{n}{4k}$ if weights are uniform.*

It is also possible to get a lower bound of $\ln(n)/2 + O(1)$ for the SPoA when weights are uniform.

Proposition 5 shows that the PoA is at least $n/4$ if weights are uniform. Here, we prove that a cost function similar to the one studied in Subsection 4.1 reaches this bound.

Proposition 6. *When the local tax function is defined as* $\frac{1}{s_{a_i}^2 + \ell_{a_i}(a)}$*, and weights are uniform, the PoA of the* SET COVER *game is at most* $\frac{n}{4} + \frac{1}{2} + \frac{1}{4n}$.

5 Concluding Remarks

The goal was to reduce the price of anarchy with particular local tax functions. However we did not consider converge issues. As a future work, we believe that our results should be completed with a study of the converge time to an equilibrium, depending on the tax function under consideration.

The model of local tax functions is fairly natural and captures several concrete situations where a coordination mechanism is introduced in order to reduce the social cost. Hence it should be fruitfully applied to other games (can it be applied to Anshelevich et al.'s connection game?). On a theoretical viewpoint, this work can also be seen as a way to understand to what extend relaxations of some conditions of the model (budget balance here) may improve the global efficiency. Hence the impact of other relaxations might also be investigated.

References

1. Koutsoupias, E., Papadimitriou, C.: Worst case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
2. Aumann, R.J.: Acceptable points in games of perfect information. Pacific Journal of Mathematics 10, 381–417 (1960)
3. Andelman, N., Feldman, M., Mansour, Y.: Strong price of anarchy. Games and Economic Behavior 65, 289–317 (2009)
4. Rosenthal, R.W.: A class of games possessing pure-strategy nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
5. Holzman, R., Law-Yone, N.: Strong equilibrium in congestion games. Games and Economic Behavior 21, 85–101 (1997)
6. Rozenfeld, O., Tennenholtz, M.: Strong and correlated strong equilibria in monotone congestion games. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) WINE 2006. LNCS, vol. 4286, pp. 74–86. Springer, Heidelberg (2006)
7. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: Proc. of FOCS 2004, pp. 295–304 (2004)
8. Epstein, A., Feldman, M., Mansour, Y.: Strong equilibrium in cost sharing connection games. In: ACM Conference on Electronic Commerce, pp. 84–92 (2007)
9. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 345–357. Springer, Heidelberg (2004)
10. Buchbinder, N., Lewin-Eytan, L., Naor, J., Orda, A.: Non-cooperative cost sharing games via subsidies. In: Monien, B., Schroeder, U.-P. (eds.) SAGT 2008. LNCS, vol. 4997, pp. 337–349. Springer, Heidelberg (2008)

Towards Network Games with Social Preferences

Petr Kuznetsov and Stefan Schmid

TU Berlin / Deutsche Telekom Laboratories, D-10587 Berlin, Germany

Abstract. Many distributed systems can be modeled as *network games*: a collection of *selfish* players that communicate in order to maximize their individual utilities. The performance of such games can be evaluated through the costs of the system *equilibria*: the system states in which no player can increase her utility by unilaterally changing her behavior. However, assuming that all players are selfish and in particular that all players have the same utility function may not always be appropriate. Hence, several extensions to incorporate also altruistic and malicious behavior in addition to selfishness have been proposed over the last years. In this paper, we seek to go one step further and study arbitrary relationships between participants. In particular, we introduce the notion of the *social range matrix* and explore the effects of the social range matrix on the equilibria in a network game. In order to derive concrete results, we propose a simplistic network creation game that captures the effect of social relationships among players.

1 Introduction

Many distributed systems have an open clientele and can only be understood when taking into account socio-economic aspects. A classic approach to gain insights into these systems is to assume that all players are selfish and seek to maximize their utility. Often, the simplifying assumption is made that all players have the same utility function. However, distributed systems are often "socially heterogeneous" whose participants run different clients and protocols, some of which may be selfish while others may even try to harm the system. Moreover, in a social network setting where members are not anonymous, some players may be friends and dislike certain other players. Thus, the state and evolution of the system depends on a plethora of different utility functions. Clearly, the more complex and heterogeneous the behavior of the different network participants, the more difficult it becomes to understand (or even predict) certain outcomes.

In this paper, we propose a more general approach to model the players' utilities and introduce a social range matrix. This matrix specifies the *perceived* costs that are taken into account by the players when choosing a strategy. For example, a player who maliciously seeks to hamper the system performance has a perceived cost that consists of the negative costs of the other players. On the other hand, an altruistic player takes into account the costs of all other players

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 14–28, 2010.

and strives for a socially optimal outcome. There are many more player types in-between that care about some players but dislike others.

In order to gain insights into the implications of different social ranges, we consider a novel network creation game that captures the willingness of a group of people to connect to each other. In this game, players do not incur infinite costs if they are not connected to some players. Rather, the utility of a player is given by the number of other players in her R-neighborhood, for some parameter R. For instance, in a game with $R = 1$, players can only collaborate with and benefit from their direct neighbors. Or imagine a peer-to-peer network like Gnutella where files are searched by flooding up to a certain radius (e.g., a time-to-live of $R = TTL = 5$), then a player is mainly interested in the data shared in her 5-hop neighborhood. Our motivation in using this model stems from its simplicity which allows to exemplify and quantify the effects of different social matrices.

1.1 Related Work

Over the last years, several models for distributed systems have been proposed that go beyond purely selfish settings. For instance, security and robustness related issues of distributed systems have been an active field of research, and malicious faults are studied intensively (e.g., [5,14]). To the best of our knowledge, the first paper to study equilibria with a malicious player is by Karakostas and Viglas [13] who consider a routing application where a single malicious player uses his flow through the network in an effort to cause the maximum possible damage. In order to evaluate the impact of such malicious behavior, a coordination ratio is introduced which compares the social costs of the worst *Wardrop equilibrium* to the social costs of the best *minimax saddle-point*. In [9], implementation problems are investigated with k faulty players in the population, but neither their number nor their identity is known. A planner's objective then is to design an equilibrium where the non-faulty players act according to her rules. Or in [1], the authors describe an asynchronous state machine replication protocol which tolerates Byzantine, Altruistic, and Rational behavior. Moscibroda et al. [17] discovered the existence of a so-called *fear factor* in the virus inoculation game where the presence of malicious players can improve the social welfare under certain circumstances. This windfall of malice has subsequently also been studied in the interesting work by Babaioff et al. [4] on congestion games.

There exists other work on game theoretic systems in which not every participating agent acts in a rational or malicious way. In the *Stackelberg theory* [19], for instance, the model consists of selfish players and players that are *controlled by a global leader*. The leader's goal is to devise a strategy that induces an optimal or near optimal so-called Stackelberg equilibrium. Researchers have recently also been interested in the effects of altruism that co-exists with selfishness [12,15]. For example, Meier et al. [15] have shown (for a specific game played on a social network) that friendship among players is always beneficial compared to purely selfish environments, but that the gain from social ties does not grow monotonically in the degree of friendship.

In contrast to the literature discussed above, we go one step further and initiate the study of games where players can be embedded in arbitrary social contexts and be selfish towards certain players, be friends with some other players, and even have enemies.

In particular, we apply our framework to a novel network creation game (for similar games, see the connection games described in Chapter 19.2 of [18]). Network creation has been a "hot topic" for several years. The seminal work by Fabrikant et al. [10] in 2003 seeks to shed light on the Internet's architecture as built by economic agents, e.g., by Internet providers or *autonomous systems*. Recent subsequent work on network creation in various settings includes [2,3,6,7,8]. Moscibroda et al. [16] considered network creation games for peer-to-peer systems. The game proposed in our paper here can be motivated by peer-to-peer systems as well. However, in contrast to [16] where peers incur an infinite cost if they are not all connected to each other, we believe that our model is more appropriate for unstructured peer-to-peer systems.

The notion of interpersonal influence matrix, similar to our social range matrices, is used in sociology for understanding the dynamics of interpersonal agreement in a group of individuals (see, e.g., [11]).

1.2 Our Contributions

The main contribution of this paper is the introduction and initial study of the social range matrix which allows us to describe arbitrary social relationships between players. For instance, social range matrices can capture classic *anarchy* scenarios where each player is selfish, *monarchy* scenarios where players only care about one network entity, or *coalitions* that seek to support players within the same coalition but act selfishly or maliciously towards other coalitions. Despite this generality, we are able to derive interesting properties of such social matrices. For instance, we show that there are matrix transformations that do not affect the equilibria points (and the convergence behavior) of a game.

In addition, as a case study, we analyze a simplistic social network creation game where players can decide to which other players they want to connect. While a new connection comes at a certain cost, a player can also benefit from her neighborhood. That is, we assume that the players' utility is given by the number of other players they are connected to up to a certain horizon, minus the cost of the links they have to pay for. For example, this game can be motivated by unstructured peer-to-peer systems where data is usually searched locally (in the peers' neighborhood) and overall connectivity is not necessarily needed. We focus on this game due to its simplicity that allows us to study the main properties of the social matrix and exemplify the concepts. For example, in a social context where players can choose their neighbors, it is likely that players will connect to those players who they are friends with. We will show that this intuition is correct and that social relationships are indeed often reflected in the resulting network. As another example, we show that the social welfare of monarchic societies can be higher than that of anarchic societies if the price of establishing a connection is relatively low; otherwise, the welfare is lower.

Our new model and the network creation game open a large number of research directions. We understand our work as a first step in exploring the effect of social ranges on the performance of network games and use this paper to report on our first insights.

1.3 Paper Organization

The rest of the paper is organized as follows. We describe our model and formally introduce the social range matrix in Section 2. Section 3 presents our first insights on the properties of a social range matrix. We then report on our case study on social network formation (Section 4). The paper is concluded with a brief discussion and an outlook on future research directions in Section 5.

2 Social Range Matrices and Perceived Equilibria

In this section, we introduce the concept of a game theory where players are embedded in a social context; in particular, we define the social range matrix F describing for each player i how much she cares about every other player j.

We consider a set Π of n players (or nodes), $\Pi = \{0, \ldots, n-1\}$. Let $\mathcal{X}_i$ be the set of possible strategies player i can pursue in a given game $\mathcal{G}$. A *strategy profile* $s \in \mathcal{X}_0 \times \ldots \times \mathcal{X}_{n-1}$ specifies a configuration, i.e., s is the vector of the strategies of all players.

The cost that actually arises at a player i in a given strategy profile s is described by its *actual* cost function $c_a(i, s)$. However, depending on the social context a player is situated in, it may experience a different *perceived cost* $c_p(i, s)$: While a purely selfish player may be happy with a certain situation, another player that cares about the actual costs of her friends may have a higher perceived cost and may want to change her strategy to a socially better one. (Note, however, that the distinction between "purely selfish players" and players that take into account the utility or cost of other players is artificial: Players whose action depends on other players' utilities can be considered "purely selfish" as well, and simply have a different cost function.)

Formally, we model the perceived costs of a given player as a linear combination of the actual costs of all other players in the game. The *social range* of player i is a vector $f_i = (f_{i0}, ..., f_{i(n-1)}) \in \mathbb{R}^n$. Intuitively, f_{ij} quantifies how much player i cares about player j, in both a positive (if $f_{ij} > 0$) and a negative way ($f_{ij} < 0$). $f_{ij} = 0$ means that i does not care about j. The social ranges of all the players constitute the *social range matrix* $F = \{f_{ij}\}$ of the game. We will later see (Lemma 1) that it is sufficient to focus on normalized matrices where $\forall i, j : -1 \leq f_{ij} \leq 1$ (rather than $f_{ij} \in \mathbb{R}$).

The perceived cost of player i in a strategy profile s is thus calculated as:

$$c_p(i, s) = \sum_j f_{ij} c_a(j, s).$$

In other words, the perceived cost of player i increases with the aggregate costs of i's *friends* (players j with $f_{ij} > 0$) and decreases with the aggregate costs of

i's *enemies* (players j with $f_{ij} < 0$). Note that we allow a player i to value other players' costs more than her own cost, i.e., f_{ii} can be smaller than some $|f_{ij}|$, $i \neq j$. This captures the effect of sacrificing one's own interests for the sake (or for the harm) of others.

Henceforth, a social matrix F with all 1's (resp., all -1's, except for f_{ii}) is called *altruistic* (resp., *malicious*). Generally, a social matrix with a lot of zero or negligibly small elements describes a system with weak social ties. Some interesting social range matrices F are:

1. If F is the identity matrix, we are in the realm of classic game theory where each player is selfish.
2. A completely altruistic scenario is described by a social matrix F consisting of 1s only, i.e., $f_{ij} = 1$ $(\forall i, j)$. Alternatively, we can also define an altruistic player that considers her own costs only to a small extent ($f_{ii} = \epsilon$, for some arbitrarily small $\epsilon > 0$).
3. In a situation where $\exists k$ such that $\forall i, j$: $f_{ij} = 0$ except for $f_{ik} = 1$, the players only care about a single individual. We will refer to this situation as a *monarchy scenario*. (Sometimes it makes sense to assume that players are at least a bit self-interested and $\forall i$: $f_{ii} = \epsilon$ for an arbitrarily small positive ϵ.)
4. If $\exists k$ such that $\forall i, j$: $f_{ij} = 0$ except for $f_{kj} = 1$ (and maybe $f_{kk} = \epsilon$), there is one benevolent player that seeks to maximize the utility of all players.
5. If $\exists k$ such that for all players i: $f_{ii} = 1$ and otherwise 0, and $f_{ki} = -1$, we have a selfish scenario with one malicious player k that seeks to minimize the utility of all the players. (Alternatively, we can also postulate that for a malicious player k, $f_{kk} = 1$.)
6. If $\exists j, k$ such that $\forall i$: $f_{ji} = f_{ki}$, then we say that players j and k *collude*: their incentives to deviate from a given strategy profile are identical. (We will show in Lemma 1 that j and k collude even if $\exists \lambda > 0$: $\forall i$, $f_{ji} = \lambda f_{ki}$.)

There are special player types to consider, e.g.:

Definition 1 (Ignorant and Ignored Players). *A player i is called* ignorant *if $f_{ij} = 0 \ \ \forall j$; the perceived cost of an ignorant player i does not depend on the actual costs. Now suppose that F contains a zero column: $f_{ji} = 0, \forall j$. In this case, no player cares about i's actual cost, and we call i* ignored.

In game theory, (pure) *Nash equilibria* are an important solution concept to evaluate the outcomes of games. A Nash equilibrium is defined as a situation where no player can unilaterally reduce her cost by choosing another strategy given the other players' strategies. In our setting, where the happiness of a given player depends on her perceived costs, the equilibrium concept also needs to be expressed in terms of perceived costs. We formally define the *perceived Nash equilibrium* (PNE) as follows.

Definition 2 (Perceived Nash Equilibrium). *A strategy profile s is a* perceived Nash equilibrium *if for every s' that differs from s in exactly one position i, we have $c_p(i, s') \geq c_p(i, s)$.*

In order to evaluate the system performance, we study the social cost of an equilibrium. Note that the social cost is defined with respect to *actual* costs: the *social cost* of a strategy profile s is defined as $\mathsf{Cost}(s) = \sum_j c_a(j, s)$. A strategy profile s is a *social optimum* if $\forall s'$: $\mathsf{Cost}(s') \geq \mathsf{Cost}(s)$.

For a given game $\mathcal{G}$ and a social matrix F, consider the ratio between the actual cost of the worst perceived Nash equilibrium and the cost of the social optimum. Comparing this ratio with the price of anarchy (the ratio computed with respect to actual Nash equilibria), we obtain the "effect of socialization" that captures the benefits or disadvantages that social relations contribute to the outcome of the game. Below we fix a game $\mathcal{G}$, and give some basic properties following immediately from the definitions.

3 Basic Properties of Social Range Matrices

We start our analysis by examining properties of the social range matrix. First, observe that F is invariant to row scaling.

Lemma 1. *Let F be a social matrix, and let $\lambda > 0$ be an arbitrary factor. Let F' be a social matrix obtained from F by multiplying a row of F by λ. Then s is a perceived Nash equilibrium w.r.t. F if and only if s is a perceived Nash equilibrium with F'.*

Proof. Let i be the player whose row is scaled. Since player i's actual costs are not affected by multiplying $f_{i\cdot}$ by λ, the perceived costs of all other players $j \neq i$ remain the same and hence, they still play their equilibrium strategy under F'. However, also player i will stick to her strategy in F':

$$c_p(i, s) = \sum_j \lambda f_{ij} c_a(j, s) \leq c_p(i, s') = \sum_j \lambda f_{ij} c_a(j, s')$$

since we know that in F, $c_p(i, s) = \sum_j f_{ij} c_a(j, s) \leq c_p(i, s') = \sum_j f_{ij} c_a(j, s')$ for all s' that differ from s in i's strategy. □

In particular, Lemma 1 implies that we can normalize a social matrix F by $f'_{ij} = f_{ij} / \max_{\ell,k} |f_{\ell k}|$.[1] Therefore, in the following, we assume normalized matrices F for which $f_{ij} \in [-1, 1]$, $\forall i, j \in \{0, \ldots, n-1\}$.

Lemma 2. *If $f_{ij} = 1 \quad \forall i, j$, then every social optimum is a perceived Nash equilibrium. If $f_{ij} = -1 \quad \forall i, j$, then every social minimum is a perceived Nash equilibrium.*

Proof. The proof is simple. By the definition of a social optimum s, $\sum_i c_a(i, s)$ is minimal, i.e., $\nexists s'$ with $\sum_i c_a(i, s') < \sum_i c_a(i, s)$. Thus, s is also an equilibrium if $f_{ij} = 1 \quad \forall i, j$, as $\nexists s'$ for a given player j with $c_p(j, s') = \sum_i c_a(i, s') < c_p(j, s) = \sum_i c_a(i, s)$.

[1] Here we assume $\max_{\ell,k} |f_{\ell k}| > 0$; otherwise, every strategy is a perceived Nash equilibrium and the price of socialization is the worst possible.

Similarly for the minimum maximizing $\sum_i c_a(i,s)$ ($\nexists s'$ with $\sum_i c_a(i,s') > \sum_i c_a(i,s)$). Profile s is also a perceived equilibrium for $f_{ij} = -1 \quad \forall i,j$, as $\nexists s'$ for a given player j with $c_p(j,s') = \sum_i c_a(i,s') > c_p(j,s) = \sum_i c_a(i,s)$. □

Note however that the opposite direction is not true: there may be games with equilibria which are not optimal, even if all players are altruistic, namely if the game exhibits local optima.

Another special case that allows for general statements are ignorant and ignored players (see Definition 1). Note that neither ignorant nor ignored players can benefit from their unilateral actions: their perceived cost functions do not depend on their strategies. Moreover, no player's perceived cost depends on the actions of an ignored player. If s is a perceived equilibrium, then any strategy s' that differs from s only in position i, where i is an ignored player, is also a perceived equilibrium. In other words, it is sufficient to determine the set of equilibria PNE' with respect to the strategies of non-ignored players Π'.

Existing literature also provides interesting results on the properties and implications of certain types of social matrices. For instance, from the work by Babaioff et al. [4]—and even earlier, from the work by Karakostas and Viglas [13]!—we know that there are games where the presence of players who draw utility from the disutility of others, can lead to an *increase* of the social welfare; this however only holds for certain game classes that are characterized by some form of a generalized Braess paradox. Or from the work by Meier et al. [15], it follows that in a virus inoculation game where the social range matrix depends on the adjacency metrics of the social network, a society can only benefit from friendship (positive entries in the social range matrix), although *not always* in a monotonic manner.

Thus, in specific game classes, some "corner case" phenomena may be observed for certain types of social matrices. In order to focus on the principal properties of the social range, in the following we concentrate on our network creation game. It turns out that in games where choosing the neighbors can be a part of a player's strategy, there is a strong correlation between the social ties and the resulting network topology.

4 Case Study: Network Creation

In this section, we give a formal definition of our network creation game and investigate the implications of different social ranges on the formed topologies.

4.1 A Network Creation Game

As a use-case for employing our game-theoretic framework, we propose a novel simple network creation game where a node (or *player*) i can decide to which other nodes j she wants to connect in an undirected graph. Establishing a connection $\{i,j\}$ (or *edge*) entails a certain cost; we will assume that connections are undirected, and that one end has to pay for it. On the other hand, a player benefits from positive network externalities if it is connected to other players

(possibly in a multi-hop fashion). We assume that the gain or cost of a player depends on the number of players in her R-hop neighborhood, for some parameter $R \geq 0$. For instance, a network creation game with $R = 1$ describes a situation where players can only benefit from (or collaborate with) their direct neighbors. As motivation for larger radii, imagine an unstructured peer-to-peer network where searching is done by flooding up to radius R, and where the number of files that can be found increases monotonically in the number of players reached inside this radius.

Formally, the actual cost of player i is given by:

$$c_a(i,s) = \alpha \cdot s_i - g(\sum_{j=1}^{R} |\Gamma^j(i,s)|)$$

where parameter $\alpha \geq 0$ denotes the cost per connection, s_i is the number of connections player i pays for, and $|\Gamma^j(i,s)|$ specifies to how many nodes node i is connected with shortest hop-distance j in a graph incurred by strategy profile s. Moreover, $g : \mathbb{N}_n \to \mathbb{R}$ is a function that specifies the utility of being in a connected group of a given size (here $\mathbb{N}_n = \{0, \ldots, n-1\}$). For example, $g(x) = x$ denotes that the utility grows linearly with the number of nodes within the given radius; a super-linear utility such as $g(x) = x^2$ may be meaningful in situations where the networking effects grow faster, and a sub-linear utility $g(x) = \sqrt{x}$ means that marginal utility of additional players declines with the size. By convention, we assume that $g(0) = 0$.

Finally, note that multiple strategy profiles (and hence perceived Nash equilibria) can describe the same network topology where the links are payed by different endpoints. Henceforth, for simplicity, we will sometimes say that a given topology *constitutes* (or *is*) a social optimum or an equilibrium if the corresponding profiles are irrelevant for the statement, are clear from the context, or if it holds for any strategy describing this network.

Given two network topologies of the same perceived costs but where one topology has some additional edges that need to be paid by a given player, this player is likely to prefer the other topology. That is, it often makes sense to assume that a player does not completely ignore the own actual cost, that is, $\forall i : f_{ii} = \epsilon$ for an arbitrarily small $\epsilon > 0$.

4.2 Social Optimum and Anarchy

First we describe the properties of the general network creation game in which players behave in a selfish manner. Social optima are characterized in the following lemma. It turns out that cliques and trees are the most efficient networks in our game.

Lemma 3. *Consider the network creation game where* $\forall x \in \mathbb{N}_{n-1}$, $g(x+1) - g(x) > \alpha/2$. *Then in the case* $R = 1$, *the only social optimum is the clique, and in the case* $R > 1$, *every social optimum is a tree of diameter at most* $\min(R, n-1)$.

Proof. Let s be any strategy profile. We say that an edge in s is *redundant* if in the strategy profile s' derived from s by dropping this edge, the R-neighborhood

of all nodes remains the same. Every non-redundant edge connecting a player with degree x to a player with degree y decreases the social cost by at least $g(x+1) - g(x) + g(y+1) - g(y) - \alpha > 0$. Naturally, every social optimum s will not have redundant edges. In the case $R = 1$, the clique has the most non-redundant edges, and thus is the only topology resulting from the social optimum.

In the case $R > 1$, suppose that the network described by s is not connected and does not contain redundant edges. Then every edge connecting the components of the graph decreases the social cost by a positive value. Hence, we can assume that the socially optimal topology is connected.

Now suppose that the network has diameter $R' > R$. Consider two nodes i and j such that j is at distance R' from i. Then adding an edge connecting i and j increases the R-neighborhood of each player by at least 1 and thus decreases the social cost. Therefore, the diameter of the social optimum topology is at most $\min(R, n-1)$.

Finally, since over all connected graphs, trees have the least number of edges and hence the cost is minimized, every social optimum results in a tree. □

In a selfish setting, players are less likely to connect to each other. Indeed, even for relatively small α, nodes remain isolated, resulting in a poor welfare.

Lemma 4. *In the network creation game, the set of isolated nodes is a Nash equilibrium if and only if* $\forall x \in \mathbb{N}_n$, $g(x) \leq x\alpha$.

Proof. Consider the strategy profile with no edges: $\forall j : s_j = 0$. If $\forall x \in \mathbb{N}_n$, $g(x) \leq x\alpha$, then unilaterally adding x edges may only increase the individual (actual) cost by at least $\alpha x - g(x)$, so no node has an incentive to deviate. On the other hand, if $\exists x \in \mathbb{N}_n$, $g(x) > x\alpha$, then every player has an incentive to add at least x edges, and thus the "isolated" strategy cannot be an equilibrium. □

Lemmas 3 and 4 imply that in the case $1 < \alpha < 2$, the cost of the social optimum in the *linear* network creation game (when $g(x) = x$) is $n(n-1)(\alpha/2 - 1)$ for $R = 1$ and $(n-1)(\alpha - 2)$ for $R > 1$, while the cost of the worst Nash equilibrium is 0, i.e., selfishness may bring the system to a highly suboptimal state.

Below we describe the conditions under which certain topologies, like cliques and trees of bounded diameter, constitute Nash equilibria of the network creation game.

Lemma 5. *In the network creation game where* $R = 1$, $\forall x \in \mathbb{N}_{\lfloor n/2 \rfloor}$, *such that* $\forall y \in \mathbb{N}_{n-x}$: $g(2x) - g(x+y) \geq \alpha(x-y)$, *every* $2x$*-regular graph constitutes a Nash equilibrium.*

Proof. Consider the strategy in which every player establishes x outgoing links so that the resulting topology is $2x$-regular. Unilaterally establishing y (non-redundant) links instead of x (for any $y \in \mathbb{N}_{n-x}$), a player pays the cost $\alpha y - g(x+y) \geq \alpha x - g(2x)$, so no player has an incentive to deviate. □

In the linear case with $R = 1$ and $\alpha < 1$, Lemma 5 implies that the clique is the only regular graph that results from an equilibrium: the only x that satisfies the

condition is $\lfloor n/2 \rfloor$. But in general, the resulting network may consist of up to $\lfloor n/2x \rfloor$ disconnected cliques of $2x$ players each.

Lemma 6. *In the network creation game with $R > 1$, where g is a monotonically increasing function on $\mathbb{N}_n$ such that $\alpha < g(n-1)$, every tree of diameter at most $\min(R, n-1)$ corresponds to a Nash equilibrium.*

Proof. Consider the strategy in which every node but one maintains one edge so that the resulting graph is a tree of diameter at least $\min(R, n-1)$. Therefore, $n-1$ players have the cost $\alpha - g(n-1)$ and one player has the cost $-g(n-1)$. Every extra edge would be redundant, and dropping edges increases the cost by at least $g(n-1) - g(n)$. Thus, no player has an incentive to deviate. □

Having described the classic setting with selfish players, we are ready to tackle social contexts.

4.3 Social Equilibria

We now turn our attention to more general matrices F, where player pairs i and j are embedded in a social context. For simplicity, we focus on values $f_{ij} \in \{-1, 0, \epsilon, 1\}$ where $f_{ij} = -1$ signifies that player i does not get along well with player j, $f_{ij} = 0$ signifies a neutral relation, and $f_{ij} = 1$ signifies friendship. We will sometimes assume that players care at least a little bit about their own cost, i.e., $\forall i : f_{ii} = \epsilon$ for some arbitrarily small positive ϵ. (This also implies that a player will not pay for a link which is already paid for by some other player.) We make two additional simplifications: we have investigated the network creation game where players can only profit from their direct neighbors (i.e., $R = 1$) in more detail, and assume a *linear* scenario where the utility of being connected to other players grows linearly in the number of contacts, that is, the marginal utility of connecting to an additional player is constant: we assume that $g(x) = x$.

Clearly, in this scenario, the cost $c_p(i, s)$ (and also $c_a(i, s)$) of a player i in a strategy profile s is independent of connections that are not incident to i. In this case, it holds that any social matrix F has a (pure) perceived equilibrium.

Lemma 7. *In the linear network creation game with $R = 1$, any social range matrix F has at least one pure perceived Nash equilibrium, for any α.*

Proof. Recall that in the $R = 1$ case, a player i can only benefit from her neighbors, that is, from a connection $\{i, j\}$ that either i or the corresponding neighbor j paid for. Player i will pay for the connection to player j if and only if the gain from this link is larger than the link cost α. We have that by establishing a new connection from player i to player j, the cost changes by $\Delta\, c_p(i) = f_{ii} \cdot (\alpha - 1) - f_{ij} \cdot 1$. If this cost is not larger than zero, it is worthwhile for player i to connect; otherwise it is not. On the other hand, player j will pay for a connection to player i iff $\Delta\, c_p(j) = f_{jj} \cdot (\alpha - 1) - f_{ji} \cdot 1 \geq 0$. As the decision of whether to connect to a given player or not does not depend on other connections, and as links cannot be canceled unilaterally, the resulting equilibrium network is unique assuming that the players will not change to a strategy of equal cost. □

Observe that the equilibrium topology found in Lemma 7 is only unique if the cost inequalities $\Delta\ c_p$ are strict. Moreover, a given equilibrium topology can result from different strategy profiles, namely if there are connections where both players have an incentive to pay for the connection to each other.

Intuitively, we would expect that the network formed by the players reflects the social context the players are embedded in. This can be exemplified in several ways. The following lemma shows that for the case of binary social matrices, there are situations where the social matrix translates directly into an equilibrium adjacency matrix.

Lemma 8. *In the linear network creation game with $R = 1$, assume a binary social matrix F where $\forall i, j : f_{ij} \in \{0, 1\}$ and where each player is aware of her own cost, i.e., $\forall i : f_{ii} > 0$. Then, for $1 < \alpha < 2$, there is an equilibrium topology that can be described by the adjacency matrix F' derived from F in the following manner: (1) $\forall i : f'_{ii} = 0$ and (2) if $f_{ij} = 1$ for some i, j, then $f'_{ij} = 1$ and $f'_{ji} = 1$.*

Proof. The claim follows from the simple observation that for $1 < \alpha < 2$, a player i is willing to pay for a connection to a player j if and only if $f_{ij} = 1$, as the cost difference is given by $\Delta\ c_p(i) = \alpha - f_{ii} - f_{ij}$: If $f_{ij} = 0$, player i only connects if $\alpha \leq 1$, and if $f_{ij} = 1$, it is worthwhile to pay for the connection as long as $\alpha \leq 2$. Therefore, as long as $1 < \alpha < 2$, one endpoint will pay for the link $\{i, j\}$ (and thus: $f'_{ij} = 1$ and $f'_{ij} = 1$) if $f_{ij} = 1$ or $f_{ji} = 1$. Clearly, it also holds that there are no loops ($\forall i : f'_{ii} = 0$). □

Note that the condition that each player cares about her own cost is necessary for Lemma 8 to hold; otherwise, if $f_{ii} = 0$, a player could trivially connect to all players as this does not entail any connection costs. In this case, the social matrix still describes a valid equilibrium adjacency matrix, however, there are many other equilibria with additional edges.

4.4 Use Case: Anarchy vs. Monarchy

A natural question to investigate in the context of social ranges is the relationship between a completely selfish society (in game theory also known as an *anarchy*) and a society with an outstanding individual that unilaterally determines the cost of the players (henceforth referred to as a *monarchy*); as already mentioned, we assume that the players always care a little bit about their own actual costs, and hence in the monarchy, let $\forall i : f_{ii} = \epsilon$ for some arbitrarily small $\epsilon > 0$, and let $\forall i : f_{ij} = 1$ where player j is the monarch (we assume $f_{ji} = 0$ for all $i \neq j$).

Interestingly, while there are situations where a monarchy yields a higher social welfare, the opposite is also true as there are settings that are better for anarchies. The following result characterizes social optima, and Nash equilibria for anarchy and monarchy settings.

Lemma 9. *In the linear network creation game with $R = 1$, the social optimum cost is $(\alpha/2 - 1)n(n - 1)$ if $\alpha < 2$ and 0 otherwise, and the anarchy has social cost $(\alpha/2 - 1)n(n - 1)$ if $\alpha \leq 1$ and 0 otherwise. For the monarchy, there can be*

multiple equilibria (of the same cost): for any α, there is always an equilibrium with cost $(\alpha-2)(n-1)$; moreover, if $\alpha \leq 1$ there is an additional equilibrium with the same cost.

Proof. We consider the social optimum, the anarchy and the monarchy in turn. *Social optimum:* If $\alpha \leq 2$, then Lemma 3 implies that any social optimum implies the clique, with the cost $(\alpha/2-1)n(n-1)$. If $\alpha > 2$, then every non-redundant link increases the social cost by $\alpha - 2$ and thus the set of isolated nodes has the minimal cost, 0.

Observe that the social cost is given by the total number of edges k in the network: k edges yield a connection cost of $k \cdot \alpha$, and the players are connected to $2k$ other players, thus $\mathsf{Cost}(s) = k \cdot \alpha - 2k$. Using Lemma 3, for the social optimum we have:

$$\min_s \mathsf{Cost}(s) = \min_k k \cdot \alpha - 2k = \begin{cases} (\alpha/2-1)n(n-1) & \text{, if } \alpha \leq 2 \text{ (clique)} \\ 0 & \text{, otherwise (disconnected).} \end{cases}$$

Anarchy: In a purely selfish setting, a player connects to another player if and only if $\alpha \leq 1$. By Lemmas 3 and 4, if $\alpha \leq 1$, then the resulting equilibrium topology is the clique and the cost is thus $(\alpha/2-1)n(n-1)$, and if $\alpha < 1$, then the resulting topology is the set of isolated nodes and the cost is 0.

$$\mathsf{Cost}(\text{Nash equilibrium}) = \begin{cases} (\alpha/2-1)n(n-1) & \text{, if } \alpha \leq 1 \text{ (clique)} \\ 0 & \text{, otherwise (disconnected).} \end{cases}$$

Monarchy: Let j denote the monarch and let $i \neq j$ denote any other player. Since the marginal utility of an additional neighbor of j is one while the connection cost is arbitrarily small ($\alpha\epsilon$), a player i will always connect (i.e., pay for the connection) to the monarch. On the other hand, the monarch will connect to a player if and only if $\alpha \leq 1$. The social cost of the network equilibrium is therefore always $(\alpha-2)(n-1)$ (up to the arbitrarily small ϵ components in the cost), for any α. □

Using Lemma 9, we can compare the efficiency of the different social settings. For relatively low connection costs, a setting with a monarch gives stronger incentives for nodes to connect, and thus socially more preferable outcomes emerge. On the other hand, for high connection costs, due to the selfless behavior of the players ignoring their own connection prices, an anarchy is preferable. As a concrete example, according to Lemma 9, for $\alpha = 3/2$, the equilibrium network of the monarchic society is a star of utility $(n-1)/2$ while in the anarchy nobody will connect, yielding a utility of zero. On the other hand, for $\alpha = 3$, the anarchy again has zero utility, while in the monarchy, players still connect which results in a negative overall utility of $-(n-1)$. Thus, the following lemma holds.

Lemma 10. *There are situations where the social welfare of anarchy is higher than the welfare in a monarchy, and vice versa.*

4.5 Windfall of Friendship and Price of Ill-Will

An interesting property of our network creation game is that more friendship relations cannot worsen an equilibrium.

Lemma 11. *Consider a social range matrix F where $\forall i,j : f_{ij} \in \{0,1\}$ and $f_{ii} = 1$. Let F' be a social range matrix derived from F where a non-empty set $\mathcal{N}$ of 0-entries in F are flipped to 1. Then, for any equilibrium strategy s^F with respect to a social matrix F, there is an equilibrium strategy $s^{F'}$ with $\mathsf{Cost}(s^{F'}) \leq \mathsf{Cost}(s^F)$.*

Proof. We prove the claim by showing that for any equilibrium strategy s^F for F, there is an equilibrium strategy $s^{F'}$ for F' that has at least as many connections as s^F. Moreover, it holds that an equilibrium with more connections always implies a higher social welfare.

First, recall from Lemma 7 that an equilibrium s^F always exists. Now fix such an equilibrium s^F from which we will construct the equilibrium $s^{F'}$. If i and j are connected in s^F, then they are still connected in $s^{F'}$: as $R = 1$, whether or not a connection $\{i,j\}$ between two players i and j is established depends on the actual costs $c_a(i,\cdot)$ and $c_a(j,\cdot)$ only. If two players i and j are not connected in s^F, they have an incentive to connect in $s^{F'}$ if $f'_{ij} = 1$ and $\alpha \leq 2$. Thus, $s^{F'}$ contains a superset of the connections in s^F. Now let k be the number of edges in a given profile s. The social cost is then given by $\mathsf{Cost}(s) = k\alpha - 2k$. For $\alpha \leq 2$, this function is monotonically decreasing, which implies the claim. On the other hand, for $\alpha > 2$, the set of isolated nodes constitutes the only equilibrium. □

Lemma 11 implies that the best equilibrium with respect to F' cannot be worse than the best equilibrium with respect to F. On the other hand, it is easy to see that a similar claim also holds for the *worst* equilibrium: Consider the equilibrium $s^{F'}$ with the fewest connections k'. Then, there is an equilibrium s^F with $k \leq k'$ edges: either $s^F = s^{F'}$, or some edges can be removed. We have the following claim.

Corollary 1. *Consider a social range matrix F where $\forall i,j : f_{ij} \in \{0,1\}$ and $f_{ii} = 1$. Let F' be a social range matrix that is derived from F by flipping one or several 0 entries to 1. Let s_w^F and s_b^F be the worst and the best equilibrium profile w.r.t. F, and let $s_w^{F'}$ and $s_b^{F'}$ be the worst and best equilibrium profile w.r.t. F' (maybe $s_w^F = s_b^F$ and/or $s_w^{F'} = s_b^{F'}$). It holds that $\mathsf{Cost}(s_w^F) \geq \mathsf{Cost}(s_w^{F'})$ and $\mathsf{Cost}(s_b^F) \geq \mathsf{Cost}(s_b^{F'})$.*

A analogous result can be obtained for settings where players dislike each other.

Lemma 12. *Consider a social range matrix F where $\forall i,j : f_{ij} \in \{-1,0\}$ and $f_{ii} = 1$. Let F' be a social range matrix derived from F where a non-empty set $\mathcal{N}$ of 0-entries in F are flipped to -1s., where $|\mathcal{N}| \geq 1$. Then, for any equilibrium strategy s^F with respect to a social matrix F, there is an equilibrium strategy $s^{F'}$ with $\mathsf{Cost}(s^F) \leq \mathsf{Cost}(s^{F'})$.*

Proof. First recall from the proof of Lemma 11 that the social welfare increases with the total number of connections given that $\forall i : f_{ii} = 1$, and that it follows from Lemma 7 that an equilibrium s^F always exists. Fix an equilibrium s^F to construct the equilibrium $s^{F'}$. Similarly to the arguments used in the proof of Lemma 11, if i and j are not connected in s^F, then they are disconnected in $s^{F'}$ as well. On the other hand, if player i pays for the connection to player j in s^F, she has an incentive to disconnect in $s^{F'}$ if $f'_{ij} = -1$ and for any non-negative α. Thus, s^F contains a superset of the connections in $s^{F'}$. □

5 Conclusions and Open Questions

We understand our work as a further step in the endeavor to shed light onto the socio-economic phenomena of today's distributed systems which typically consist of a highly heterogeneous population. In particular, this paper has initiated the study of economic games with more complex forms of social relationships. We introduced the concept of social range matrices and studied their properties. Moreover, we have proved the intuition right (under certain circumstances) that in our novel network creation game, the social relationships are reflected in the network topology.

This paper reported only on a small subset of the large number of questions opened by our model, and we believe that there remain many exciting directions for future research. For instance, it is interesting to study which conditions are necessary and sufficient for counter-intuitive phenomena such as the fear factor and the windfall of malice [4,17], or the non-monotonous relationship between welfare and friendship in social networks [15]. Another open question is the characterization of all topologies that correspond to a Nash equilibrium.

References

1. Aiyer, A., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., Porth, C.: BAR Fault Tolerance for Cooperative Services. In: Proc. 20th ACM Symposium on Operating Systems Principles (SOSP), pp. 45–58 (2005)
2. Albers, S., Eilts, S., Even-Dar, E., Mansour, Y., Roditty, L.: On Nash Equilibria for a Network Creation Game. In: Proc. 17th ACM Symposium on Discrete Algorithms, SODA (2006)
3. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. In: Proc. 45th Symposium on Foundations of Computer Science (FOCS), pp. 295–304 (2004)
4. Babaioff, M., Kleinberg, R., Papadimitriou, C.: Congestion Games with Malicious Players. In: Proc. ACM Conference on Electronic Commerce (EC), San Diego, CA, USA (2007)
5. Castro, M., Liskov, B.: Practical Byzantine Fault Tolerance. In: Proc. 3rd Symposium on Operating Systems Design and Implementation (OSDI), pp. 173–186 (1999)
6. Chen, H.-L., Roughgarden, T.: Network Design with Weighted Players. In: Proc. 18th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 29–38 (2006)

7. Corbo, J., Parkes, D.C.: The Price of Selfish Behavior in Bilateral Network Formation. In: Proc. 24th ACM Symposium on Principles of Distributed Computing (PODC), pp. 99–107 (2005)
8. Demaine, E.D., Hajiaghayi, M.T., Mahini, H., Zadimoghaddam, M.: The Price of Anarchy in Network Creation Games. In: Proc. 26th Annual Symposium on Principles of Distributed Computing (PODC) (2007)
9. Eliaz, K.: Fault Tolerant Implementation. Review of Economic Studies 69, 589–610 (2002)
10. Fabrikant, A., Luthra, A., Maneva, E., Papadimitriou, C.H., Shenker, S.: On a Network Creation Game. In: Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC), pp. 347–351 (2003)
11. Friedkin, N.E., Johnsen, E.C.: Social Influence Networks and Opinion Change. Advances in Group Processes 16 (1999)
12. Hoefer, M., Skopalik, A.: Altruism in Atomic Congestion Games. In: Proc. European Symposium on Algorithms (ESA) (2009)
13. Karakostas, G., Viglas, A.: Equilibria for Networks with Malicious Users. Mathematical Programming A 110(3), 591–613 (2007)
14. Li, H., Clement, A., Marchetti, M., Kapritsos, M., Robinson, L., Alvisi, L., Dahlin, M.: FlightPath: Obedience vs Choice in Cooperative Services. In: Proc. Symposium on Operating Systems Design and Implementation, OSDI (2008)
15. Meier, D., Oswald, Y.A., Schmid, S., Wattenhofer, R.: On the Windfall of Friendship: Inoculation Strategies on Social Networks. In: Proc. 9th ACM Conference on Electronic Commerce (EC) (2008)
16. Moscibroda, T., Schmid, S., Wattenhofer, R.: On the Topologies Formed by Selfish Peers. In: Proc. 25th Annual Symposium on Principles of Distributed Computing, PODC (2006)
17. Moscibroda, T., Schmid, S., Wattenhofer, R.: When Selfish Meets Evil: Byzantine Players in a Virus Inoculation Game. In: Proc. 25th Annual Symposium on Principles of Distributed Computing (PODC) (2006)
18. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.: Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
19. Roughgarden, T.: Stackelberg Scheduling Strategies. In: Proc. 33rd ACM Symposium on Theory of Computing (STOC), pp. 104–113 (2001)

Distributed Weighted Stable Marriage Problem

Nir Amira, Ran Giladi, and Zvi Lotker

Department of Communication Systems Engineering
Ben-Gurion University Beer-Sheva 84105 Israel
{niramir,ran,zvilo}@bgu.ac.il

Abstract. The Stable Matching problem was introduced by Gale and Shapley in 1962. The input for the stable matching problem is a complete bipartite $K_{n,n}$ graph together with a ranking for each node. Its output is a matching that does not contain a blocking pair, where a blocking pair is a pair of elements that are not matched together but rank each other higher than they rank their current mates. In this work we study the Distributed Weighted Stable Matching problem. The input to the Weighted Stable Matching problem is a complete bipartite $K_{n,n}$ graph and a weight function W. The ranking of each node is determined by W, i.e. node v prefers node u_1 over node u_2 if $W((v,u_1)) > W((v,u_2))$. Using this ranking we can solve the original Stable Matching problem. We consider two different communication models: the billboard model and the full distributed model. In the billboard model, we assume that there is a public billboard and each participant can write one message on it in each time step. In the distributed model, we assume that each node can send $O(\log n)$ bits on each edge of the $K_{n,n}$. In the billboard model we prove a somewhat surprising tight bound: any algorithm that solves the Stable Matching problem requires at least $n-1$ rounds. We provide an algorithm that meets this bound. In the distributed communication model we provide an algorithm named *intermediation agencies algorithm*, in short (IAA), that solves the Distributed Weighted Stable Marriage problem in $O(\sqrt{n})$ rounds. This is the first sub-linear distributed algorithm that solves some subcase of the general Stable Marriage problem.

Keywords: Stable Marriage, Distributed Algorithms, Matching, Billboard, Scheduling.

1 Introduction

The Stable Marriage (or Stable Matching) problem was first introduced by Gale and Shapley in [1]. In general, stable marriage is a matching criterion on a complete bipartite graph $G(V = In \bigcup Out, E) = K_{n,n}$, where In and Out are the two sides of the bipartite graph G and $E = \{(i,j) : i \in In, j \in Out\}$. The two sides can take the role of Boys&Girls, Hospitals&Students, Input ports&Output ports, and more. Each node $v \in In$ has a ranking for the Out nodes and each node $u \in Out$ has a ranking for the In nodes. The goal is to find a stable match between the In and Out nodes using the preferences of all nodes. We say that a

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 29–40, 2010.

stable match is a match that does not include blocking pairs, where a blocking pair is a pair of elements that are not matched together but rank each other higher than they rank their current mates. This situation leads to a possible betrayal, where such an element in the match will have reason to change the match. It should be noted that there could be many stable matches for a specific problem, some prioritizing one side's preferences over the other's, while some are more balanced (or "fair").

There is extensive research on stable marriage algorithms, covering combinatorial as well as other methods, distributed and centralized algorithms, fairness and performance considerations, etc. Gusfield and Irving in [2] present a large number of variants on the original problem, and the known algorithms for them. A fair solution using genetic algorithms for the Stable Marriage problem was offered in [3]. Distributed Stable Matching problems with ties and incomplete lists were suggested by Brito Meseguer [4]. An iterative parallel approach was discussed in [5], suggesting an improved algorithm in terms of speeding up the convergence process. Feder et al. [6] present a PRAM algorithm based on linear programming methods that find a stable matching in $O^*(\sqrt{m})$ time by a polynomial number of processors, where m is the total length of preference lists of individuals. Kipnis and Patt-shamir in [7] analyzed the distributed complexity of the Stable Marriage problem. They prove a lower bound of $\Omega(\sqrt{n/B \log n})$ when B is the number of bits per message, and provide an algorithm that solves the distributed stable marriage in $O(D + |E|)$ when D is the graph diameter. Use of Stable Marriage algorithms in network devices was suggested by Chuang et al. [8], for a CIOQ switch with a speed-up of 2, and they proved that this kind of implementation works like an OQ switch (which is optimal).

The motivation of the current paper is derived from [7] and [8]. In order to reach a better understanding we study the following simpler case of the general Stable Marriage problem. We assume that each edge $e \in E$ has a weight $W(e)$ and that both sides want to maximize their weights. This problem variant is suitable when both sides want to cooperate. We call this problem the *Weighted Stable Marriage Problem*, or WSM. One application of this model is load balancing in a VOQ switch. A Virtual Output Queues switch (VOQ switch) is a network element with input and output ports. Each input port maintains a separate queue for each output port. The queues store the incoming data cells until the cells are transferred to the output ports. The decision of which cell to transfer in each phase is made by a scheduling algorithm. The interested reader can see [9,10] for more details. The In side in G is the input ports of the VOQ switch and the Out side is the output ports. The edge weights are determined by the number of packets in each queue in the input ports that destined to each output port. We believe that the use of a WSM-based algorithm as a scheduling algorithm can achieve better results than the current highest level schedulers such as iSlip [9].

Our results. There is a big gap in our understanding of distributed stable marriage. In this work we take the first step towards closing this gap. We study

Weighted Stable Matching in two different communication models: the billboard model and the full distributed model (Section 2).

We prove a tight bound of $O(n)$ in the billboard model (Section 3). We show that any algorithm that solves the Stable Matching problem requires at least $n-1$ steps. We provide an algorithm that solves the Stable Matching problem in $n-1$ steps; and any greedy algorithm solves the problem at time n. This billboard model lower bound holds also for the original, more general problem of stable marriage with two sides preferences.

We present the first sub-linear distributed algorithm on the full distributed weighted stable marriage model. Our algorithm IAA (Section 4) solves the problem in $O(\sqrt{n})$ steps.

2 Model and Notation

In this section, first we provide basic notation for our models, and then we formally present the models. Let $[n] := \{1, ..., n\}$ be the set of all integers between 1 and n. To keep the algorithm simple we will assume that $\sqrt{n} \in \mathbb{N}$ is an integer.

Edges in all the models can be classified into three types: Matched, Forbidden and Valid. Assume we are looking at an algorithm at time t. Matched edges $M(t) \subset E$ are the set of edges that connect two matched nodes; Forbidden edges $F(t) \subset E$ have at least one of their nodes already matched; and Valid edges $V(t) \subseteq E$ are edges none of whose nodes is matched, so they can still be part of the matching. The sets are a non-overlapping partitions of E, i.e. at time t each edge $e \in E$ is in one and only one of these sets. An edge can move only from the Valid set to the Forbidden or Matched set.

In the Stable Marriage bipartite graph one side will be called In and the other Out. The bipartite graph will be marked as an $N \times N$ table or game board, where rows are the In nodes, columns are the Out nodes and edges are the table slots. The time units of any algorithm are called steps or rounds, and will be marked by R. We will handle only the case of equal number of input and output ports and leave for future work the unequal case.

For convenience reasons we studied the case of unique preferences lists with no ties. The case of preferences with ties can be derived from the unique case using a tiebreaker method. The following notation for the preferences lists is given. Let the vector $Pref_v$ be the ordered preferences list of node $v \in In$, so $\forall i \in [n]$; $Pref_v(i)$ value is the output port that input port v prefer in position i.

A complete description of any variant model of the Stable Marriage problem is composed of two sub-models: The Communication Model and the Preferences Model. The Communication Model defines how nodes send messages to one another and the Preferences Model describes which and in what order nodes hold preferences on the nodes of the other side.

2.1 Communication and Preferences Models

Preferences Models. The default and well-known Preferences Model for the Stable Marriage problem is the Two-Sided Model, where each node of each side

ranks all the nodes in the other side with no ties. There are many variations to this model, such as using incomplete preferences lists or by including ties in the preferences lists. These variants and more are covered in [2]. Another variant is using master lists [11], such that all node preferences of one side of the graph follow a master list that dictates their preference order.

In this paper we focus on a simple model called the Weighted Model in which there are no private preferences lists to each side but each edge has a unique weight, and by ordering the weights of all the edges connected to a node $v \in V$ the node can generate its own preference list in the following way. Input port $i \in In$ prefers output port $j_1 \in Out$ over output port $j_2 \in Out$ if $W((i, j_1)) > W((i, j_2))$. We use the same criterion on the output ports, i.e. we say that output port $i \in Out$ prefers input port $j_1 \in In$ over input port $j_2 \in In$ if $W((i, j_1)) > W((i, j_2))$. We can justify this preference order using our VOQ switch motivation: we are interested in minimizing the size of the queues hence we prefer matching bigger queues (bigger weights).

Communication Models. In this paper we use two communication models. The first communication model we use is the Distributive Model which is applied in a growing world of parallel processors. In this model each user is an individual processing unit with different communication channels to each of the other users. The main advantage of distributive methodology is that it enables building safer, more robust and efficient systems than in centralized computing.

In this Distributive Model each edge is a bidirectional communication link that can transfer $O(\log n)$ bits in each message and each node in each step can send a different message on each edge. In Section 4 we provide an algorithm for the Distributive Model.

The second model we use is the Public Billboard model in which all communication among the nodes is performed through a public billboard such that all nodes can view all messages. It is a well-studied model in recommendation systems, see [12,13,14,15]. In this model users try to recommend items. A user can test an item, and then write his recommendations on the billboard such that all other users can read it. The goal is to fairly reduce the user's total work using the billboard.

In the original billboard model, in each algorithm step each node can publish a message of $O(\log n)$ bits constructed from a random combination of all the data it owns. We restrict this model and let each node publish only one edge value it owns to the billboard. When using this model it is easy to define the problem with cards in the following way. We refer to the table slots in the game board as cards that can face up and be shown to all, or face down (see figure 1). As in the table slots the value that $card_{ij}$ in row i and column j hold is of the edge between node i and j. (if two sided preferences model is used then two cards are needed in each slot to cover directed edges). When we say, "publish on a public billboard" we mean "flip the appropriate game card". In section 3 we prove a tight bound for the stable marriage problem in the Billboard Model.

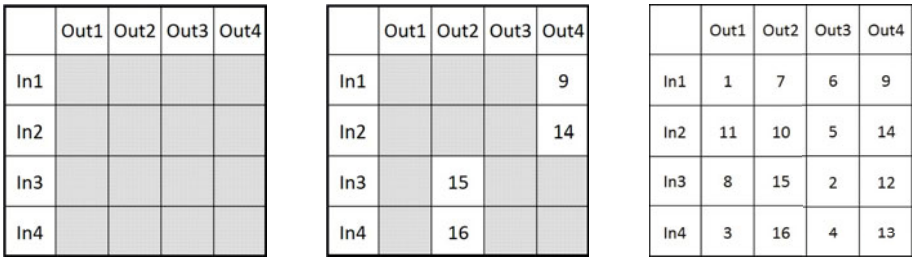


Fig. 1. (a) A table of cards facing down. (b) Each row flipped its maximal card and now the maximal cards are facing up. (c) All the cards are facing up.

2.2 Basic Properties of the Distributive Weighted Model

In this subsection we focus on a model composed of the weighted preferences sub-model with the distributive communication sub-model, We name it the Distributive Weighted model. We will show some properties. The first simple lemma shows that the maximum weighted edge is in the stable matching.

Lemma 1. *If the weights are unique the maximum value edge in the bipartite graph will always be part of the stable matching M.*

Proof. Let e be the edge that has maximum weight i.e. $W(e) > W(e')$ for all $e' \neq e \in E$. By contradiction let us say that there exists a stable matching M such that $e \notin M$. Let in_{max} and out_{max} be the nodes connected to edge e from both sides. Since e is not in the final match, in_{max} and out_{max} are not matched together but obviously they prefer each other more than anyone else; therefore $\{in_{max}, out_{max}\}$ is a blocking pair and there is a crossover, hence M is not stable and we have a contradiction. □

The next lemma follows from the previous one.

Lemma 2. *In any Weighted Stable Marriage problem, if the weights are unique there is only one stable match.*

Proof. From lemma 1 it follows that any algorithm must insert the maximum weight edge e to M, then all edges $N(e)$, the neighborhood of e, cannot be in the matching so we update the Forbidden set to be $F = F \cup N(e)$, and the Valid set accordingly, $V = V \setminus e \cup N(e)$. It follows that in V there will not be edges connected to nodes in_{max} or out_{max}, meaning we reduced the graph to an (n-1)-bipartite graph. Because e is unique, the bipartite graph is also unique. Applying this procedure by continuously using lemma 1 to find the maximum node on the next bipartite graph will result in finding a unique Stable Marriage. □

Looking closely at the above proof we can generate a simple solution to the problem by using the Gale and Shapley algorithm in the following way: each In port sends a proposal to the output port of its maximal edge weight, and the

Out ports reject or accept each proposal. We repeat these two steps until all *Out* ports receive a proposal. Hoepman [16] presents a similar distributed algorithm that computes a weighted matching at most a factor 2 away from the maximum.

In section 4 we will present an improved algorithm for the Distributive Weighted model that works in $\sqrt{n}$ steps.

3 Public Billboard Tight Bound

In this section we prove a tight bound of $n-1$ on the Public Billboard model. First we present a lower bound optimal algorithm named *Maximal Card algorithm*. Next we prove our lower bound: any algorithm needs at least $n-1$ time steps to solve the Stable Marriage problem in the Billboard model.

Definition 1. *In the Maximal Card algorithm, in each step the card that each node chooses to flip is its next favorite card from the Valid list.*

We will prove that the worst case running time of this algorithm is $n-1$. The proof here is similar to the proof of Lemma 2, in each step we insert the maximum value edge to the matching and decreases the game to $(n-1) \times (n-1)$; after $n-1$ steps we are left with a 1×1 game and the match is obvious. This worst case is our $n-1$ upper bound. In this algorithm each edge is inserted to the matching set only when there is no doubt this edge belongs to the matching, hance there is no need to use backtracking like it is used in the original Gale & Shapley algorithm.

In addition we show that the improvement of *Maximal Card algorithm* over any other algorithm can be only one step.

Lemma 3. *After n steps any greedy algorithm will find a stable matching in the Public Billboard model.*

Proof. By "greedy algorithm" we mean that all nodes submit new data at each step, so that at each step n more cards are visible to all. The explanation for this lemma is simple: after n steps of any greedy algorithm, all the cards are facing up and all the nodes can see all the game board; each node can calculate the matching locally. □

3.1 Public Billboard Lower Bound

The next theorem says that any algorithm that uses the Public Billboard model requires at least $n-1$ time steps. In order to state the theorem we need the following definition. Let $\tau(i)$ be the first time there is at least i matches in M, $|M(\tau(i))| \geq i$, i.e.

$$\tau(i) = min\{t : |M(t)| \geq i\}$$

As in the case of *Maximal Card algorithm* we can say that any algorithm that uses the Public Billboard model can insert edges to the matching only when there is no doubt edges belong to the matching, hance we don't use backtracking in any algorithm.

Theorem 1. *For any* $i \in [n-1]$, $\tau(i) \geq i$.

Proof. We now look at a specific input for the Public Billboard matching problem. Assume $W((i,j)) > W((i,j+1))$ for all $i \in [n]$, $j \in [n-1]$. This means that all the In ports prefer the output ports by order i.e. the first output port over the second, the second over the third and so on. We claim that in order to match an In port to the i-th output port we need to know at least $n-i+1$ weights. We prove this by induction on i. Clearly for $i=1$ we need to know all the n weights of the edges from the In ports to the first output port, otherwise if we try to take a matching decision without knowing all the n weights it is possible that we don't know the maximal weight hence the matching decision will be wrong. Now assume that the claim holds for $i=k$, we need to prove that the claim holds for $i=k+1$. The first k output ports have priority over the $k+1$ output port, so assuming we know the matching of the first k output ports, we can ignore the In ports that are matched to them. But the rest $n-k$ of the In ports must reveal their maximal value on the billboard, otherwise it is possible that we have not revealed the maximal one. Therefore, in order to match $|M(\tau(i))|$ pairs in the Billboard Model the number of weights we need to reveal is at least:

$$\sum_{i=1}^{|M(\tau(i))|} (n-i+1) = \frac{|M(\tau(i))|}{2}\left(2n - |M(\tau(i))| + 1\right) \tag{1}$$

Note that the above equation is not true for the last round; in fact if we know the first $n-1$ matched pairs we also know the last one.

Since we have a specific preference for the input ports it follows that once an input port is matched its weights are no longer relevant. Therefore the number of weights that are not relevant at time $\tau(i)$ is:

$$\sum_{i=1}^{|M(\tau(i))|} (n-i) = \frac{|M(\tau(i))|}{2}\left(2n - |M(\tau(i))| - 1\right) \tag{2}$$

In each step in the Billboard Model we can reveal n weights, hence if we have $\tau(i)$ steps we can reveal $n\tau(i)$ weights. And so the number of relevant weights we can reveal is:

$$n\tau(i) - \frac{|M(\tau(i))|}{2}\left(2n - |M(\tau(i))| - 1\right) \tag{3}$$

This number should be bigger than the number of weights we need to reveal that we computed in (1) and so we get the following inequality:

$$n\tau(i) - \frac{|M(\tau(i))|}{2}(2n - |M(\tau(i))| - 1) \geq \frac{|M(\tau(i))|}{2}\left(2n - |M(\tau(i))| + 1\right) \tag{4}$$

We can write the above inequality simply as:

$$n\tau(i) \geq (2n - |M(\tau(i))|)|M(\tau(i))| \tag{5}$$

We know that $i \leq n$ and from lemma 3 we know that it is possible to solve the Stable Matching problem in n time steps. Therefor we can assume that $M(\tau(i)) \leq n$; in this case, we can use the definition of $|M(\tau(i))| > i$ and we get that:

$$n\tau(i) \geq (2n - |M(\tau(i))|)|M(\tau(i))| \geq (2n - i)i \tag{6}$$

Now we can use $i \leq n$ again and simplify the previous inequality:

$$n\tau(i) \geq (2n - n)i \tag{7}$$

and so we get that:

$$\tau(i) \geq i \tag{8}$$

This concludes the proof. □

4 Algorithm for the Distributive Weighted Model

According to Lemma 2 there is just one stable matching per instance, so this algorithm needs to find one stable matching and this matching will be the only one.

We will introduce the term *Intermediation Agencies* (IA); these are $\sqrt{n}$ arbitrary elements from the set of Out ports such that each is responsible for a set of preferences from the In side. $\forall i \in [\sqrt{n}]$, IA_i is responsible for the $\sqrt{n}$ preferences in positions $\{(i-1)\sqrt{n}+1, ..., i\sqrt{n}\}$ of the In ports. The algorithm name is according to the IA, *Intermediation Agencies Algorithm*, or IAA. (See an example for the IAA algorithm in figure 2)

In this section $M(i)$ is the set of edges that are in the stable matching at time i where i is the number of rounds from the second phase and not from the beginning of the algorithm. Let $Val_v(i)$ $\forall i \in [n]$ be the value of the edge from node $v \in In$ to node $Pref_v(i)$.

This is a synchronized algorithm, which means that all the nodes follow a common clock. It is constructed from two phases and we will describe each of them in detail below.

Phase 1: The objective of this phase is to move all the data (n^2 values) from the In ports to the IAs (each IA receives $n\sqrt{n}$ values). All the In ports simultaneously send to IA_1 their first choice, to IA_2 their $\sqrt{n}+1$ choice and so on. In one step, each input port sends $\sqrt{n}$ values. In general we can write that in each step $R \in [\sqrt{n}]$ and $\forall i \in [\sqrt{n}]$ all In ports simultaneously send to IA_i their $(i-1)\sqrt{n}+R$ choice.

Phase 2: This phase is designed to inform all the nodes about the stable solution. The IAs perform a local centralized calculation one after the other (using the Maximal card algorithm or the Gale & Shapley algorithm) and find the weighted stable matching. Each IA knows its position hence it knows when to compute the next matched couples (minimum of $\sqrt{n}$ couples). In step $2i-1$ from the beginning of Phase II, IA_i calculates a matching and informs all the matched input ports who is their match; in the sequential step all the matched input

Algorithm 1. IAA

Phase I - Move Phase
for $R = 1, \sqrt{n}$ **do**
  for all $i \in [\sqrt{n}]$ & $v \in IN$ **do** ▷ simultaneously all the In ports send
    $IA_i \leftarrow \{Pref_v((i-1)\sqrt{n}+R), Val_v((i-1)\sqrt{n}+R)\}$ ▷ their preferences
  end for ▷ to all the IA
end for

Phase II - Inform Phase
$i \leftarrow 1$
while $|M| < n$ **do** ▷ sequentially each IA adds edges to the matching
  IA_i calculate local Matching
  IA_i informs all the matched nodes from the In set on their match
  the matched In inform their match and all the IAs
  each IA updates its local $V(i), M(i), F(i)$
  $i++$
end while

ports inform their matches and all the IAs that they are matched. Now each IA update its $V(t)$, $F(t)$ and $M(t)$ lists so it can know which edges can still be matched and which are Forbidden.

Time Complexity: In order to move all the n^2 values of the edges to the IAs, the In ports send at each step one value to each IA. There are $\sqrt{n}$ IAs, hence in one step all the In ports send $n\sqrt{n}$ values together, so that $\sqrt{n}$ steps are needed to transfer all the data to the intermediation agencies. In the second phase, the IAs are working sequentially and for each IA two steps are required, hence we need $2\sqrt{n}$ steps. In total the time complexity of this algorithm is $O(\sqrt{n})$.

Correctness: In order to show that at the end of this algorithm each port knows who is its match we suggest the following lemma:

Lemma 4. *After Phase I, IA_1 holds at least $\sqrt{n}$ values that belong to M.*

Proof. During the first phase, each port sends one edge every time step, i.e. IA_1 receive $n\sqrt{n}$ edges. Now IA_1 can perform locally the Gale & Shapley algorithm. Since each In port sends its highest $\sqrt{n}$ preferences to IA_1 it follows that $b \geq \sqrt{n}$ Out ports will receive a proposal at the proposal phase of Gale & Shapley algorithm, and this proposal cannot be overcome in the next steps by other IAs since the next IA holds only lower weights. This means that b edges will be added to $M(1)$, the Matched set, and the rest to $F(1)$, the Forbidden set. □

Theorem 2. *Algorithm IAA computes a stable matching.*

Proof. The proof is by induction on the number of input ports in the bipartite graph. Clearly if $|In|$ is 1,2 or 3, IAA algorithm returns a stable matching since there is only one IA and it holds all the game weights.

For the inductive step, assume that when the number of In ports is m IAA returns a stable matching. We have to show that IAA returns a stable matching

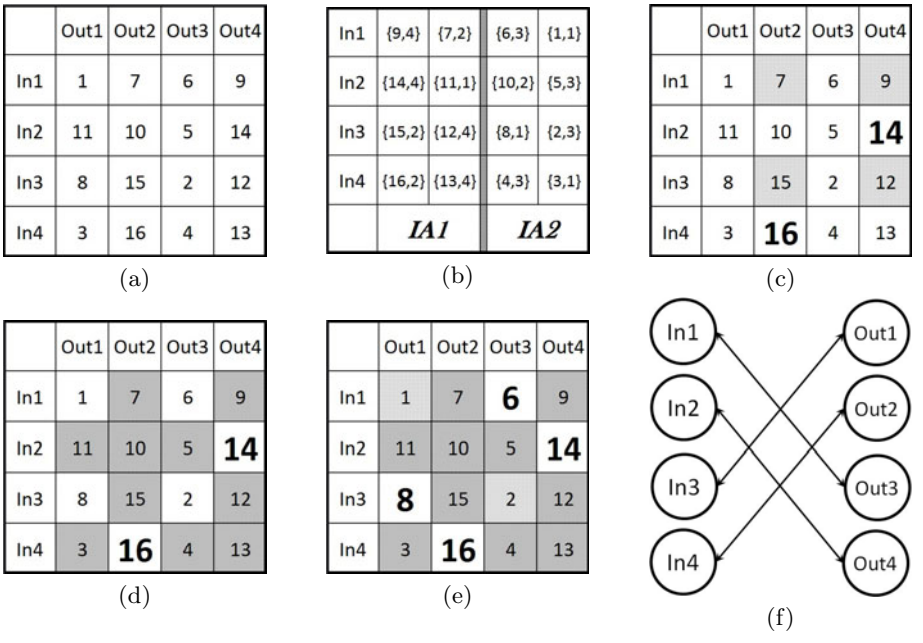


Fig. 2. An example for the IAA algorithm (a) This table represents the initial complete bipartite graph with weights on the edges. From these weights each node builds its own preference list, e.g. $in1$ preference list is: $\{out4, out2, out3, out1\}$ because $9 > 7 > 6 > 1$. (b) The data each IA gets from the In ports after Phase I. The data are ordered by weight and each slot is made of $\{Weight, OutPort\#\}$. (c) In Phase II - After $IA1$ informs all the In ports about the matching. The bold slots (16 and 14) are the matched slots and the gray slots represent the unsuccessful attempts of $IA1$ to match. (d) In Phase II - After each node updates its local variables. The gray slots represent the values that moved to the forbidden set. (e) In Phase II - After $IA2$ informs all the In ports about the matching. This is the end of the algorithm since all the ports are matched. (f) The output of the algorithm - a Stable Matching.

if the number of In ports is $m+1$. From lemma 4 we know that at the first step of *Phase II* the IAA algorithm computes the stable set $M(1)$ and that the size of $M(1)$ satisfies $|M(1)| \geq \sqrt{m+1}$ [1]. In *Phase II*, let k be an integer such that $1 < k \leq \sqrt{m+1}$, we assume that $i = k$ and that $|M(i-1)| \geq (i-1)\sqrt{m+1}$. Now note that there are $m + 1 - |M(i-1)|$ unmatched In ports. We continue by case analysis.

First assume that:

$$m + 1 - |M(i-1)| > \sqrt{m+1}$$

[1] Clearly both $\sqrt{m}$ and $\sqrt{m+1}$ cannot be an Integer if $m > 1$, this technical problem can be solved by rounding up the number of edges each IA get and rounding down the number of IAs, for sake of simplicity we ignore this case.

From *Phase I* we know that each In port sent $\sqrt{m+1}$ different edges to IA_i, so particulary the unmatched In ports sent. So it follows that at step i of *Phase II* $|M(i)| \geq |M(i-1)| + \sqrt{m+1}$. Moreover we can apply a similar argument as we did in lemma 4 and obtain that the set $M(i)$ is stable. Therefore it follows that there exists $1 \leq i \leq \sqrt{m+1}$ such that the number of unmatched ports satisfies $m+1-|M(i-1)| \leq \sqrt{m+1}$. Now we can proceed to the second case in less then an order of $\sqrt{n}$ rounds.
The second case is:

$$m+1-M(i-1) \leq \sqrt{m+1}$$

in this case IA_i knows all the remaining edges and so it can compute all the missing edges in the stable matching, and the algorithm is finished after a constant number of steps. □

5 Conclusions and Open Problems

Stable Marriage is a natural model that describes equilibrium. We believe that understanding the distributed time complexity of the Stable Marriage problem is a central question.

In this paper we concentrate on a simple version of Weighted Stable Marriage. It is possible to show that this problem is equivalent to the master list stable marriage. We study this problem in two different models. In the Billboard Model we show a tight bound, i.e. any algorithm that solves the stable matching problem requires at least $n-1$ steps, we also provide such an algorithm. In the Distributive Weighted model we provide an algorithm IAA that solves the Stable Marriage problem in $O(\sqrt{n})$ steps. We remark that all the lower bounds that appear in [7] can work in the case that the preferences lists are derived from the edges weights.

There are still many open problems such as what is the time complexity of the distributed stable marriage problem in the two sided preferences model? Is the (n-1) Lower bound of the billboard model valid also in the case where arbitrary data can be written at each round?

There are many different models to explore. For example: in a bipartite graph each edge has a weight. The In ports determine their ranking according to the maximum weights while the Out ports determine theirs according to the minimum weights. This model captures a competitive market. On the other hand, the problem we study in this paper captures cooperation between the In and Out ports.

The possibility of using the Stable Marriage criteria in a VOQ switch is mentioned in the paper and in this field a performance analysis of the Stable Marriage criteria as a packet scheduling is needed.

Acknowledgments

We would like to thank the anonymous reviewers for very constructive and detailed comments.

References

1. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. The American Mathematical Monthly 69, 9–15 (1962)
2. Gusfield, D., Irving, R.W.: The stable marriage problem: structure and algorithms. MIT Press, Cambridge (1989)
3. Nakamura, M., Onaga, K., Kyan, S., Silva, M.: A genetic algorithm for sex-fair stable marriage problem. In: ISCAS, pp. 509–512 (1995)
4. Brito, I., Meseguer, P.: Distributed stable matching problems with ties and incomplete lists. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 675–679. Springer, Heidelberg (2006)
5. Lu, E., Zheng, S.Q.: A parallel iterative improvement stable matching algorithm. In: Pinkston, T.M., Prasanna, V.K. (eds.) HiPC 2003. LNCS (LNAI), vol. 2913, pp. 55–65. Springer, Heidelberg (2003)
6. Feder, T., Megiddo, N., Plotkin, S.A.: A sublinear parallel algorithm for stable matching. In: SODA 1994: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics, pp. 632–637 (1994)
7. Kipnis, A., Patt-Shamir, B.: A note on distributed stable matching. In: ICDCS, pp. 466–473. IEEE Computer Society, Los Alamitos (2009)
8. Chuang, S., Goel, A., McKeown, N., Prabhakar, B.: Matching output queueing with a combined input output queued switch. Technical report, Stanford, CA, USA (1998)
9. McKeown, N.: The islip scheduling algorithm for input-queued switches. IEEE/ACM Trans. Netw. 7, 188–201 (1999)
10. Banovic, D., Radusinovic, I.: Scheduling algorithm for voq switches. AEU - International Journal of Electronics and Communications 62, 455–458 (2008)
11. Irving, R.W., Manlove, D.F., Scott, S.: The stable marriage problem with master preference lists. Discrete Applied Mathematics 156, 2959–2977 (2008)
12. Awerbuch, B., Patt-Shamir, B., Peleg, D., Tuttle, M.: Collaboration of untrusting peers with changing interests. In: Proceedings of the 5th ACM conference on Electronic commerce, pp. 112–119. ACM, New York (2004)
13. Awerbuch, B., Hayes, T.P.: Online collaborative filtering with nearly optimal dynamic regret. In: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures, pp. 315–319. ACM, New York (2007)
14. Awerbuch, B., Kleinberg, R.: Competitive collaborative learning. J. Comput. Syst. Sci. 74, 1271–1288 (2008)
15. Awerbuch, B., Azar, Y., Lotker, Z., Patt-Shamir, B., Tuttle, M.R.: Collaborate with strangers to find own preferences. In: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures, pp. 263–269. ACM, New York (2005)
16. henk Hoepman, J.: Simple distributed weighted matchings. In eprint cs.DC/0410047 (2004)

Traffic Grooming in Star Networks via Matching Techniques*

Ignasi Sau[1], Mordechai Shalom[2], and Shmuel Zaks[1]

[1] Department of Computer Science, Technion, Haifa, Israel
{ignasi,zaks}@cs.technion.ac.il
[2] TelHai Academic College, Upper Galilee, 12210, Israel
cmshalom@telhai.ac.il

Abstract. The problem of grooming is central in studies of optical networks. In graph-theoretic terms, it can be viewed as assigning colors to given paths in a graph, so that at most g (the *grooming factor*) paths of the same color can share an edge. Each path uses an ADM at each of its endpoints, and paths of the same color can share an ADM in a common endpoint. There are two sub-models, depending on whether or not paths that have the same color can use more than two edges incident with the same node (*bifurcation allowed* and *bifurcation not allowed*, resp.). The goal is to find a coloring that minimizes the total number of ADMs. In a previous work it was shown that the problem is NP-complete when bifurcation is allowed, even for a star network. In this paper we study the problem for a star network when bifurcation is not allowed. For the case of simple requests - in which only the case of $g = 2$ is of interest - we present a polynomial-time algorithm, and we study the structure of optimal solutions. We also present results for the case of multiple requests and $g = 2$, though the exact complexity of this case remains open. We provide two techniques, which lead to $\frac{4}{3}$-approximation algorithms. Our algorithms reduce the problem of traffic grooming in star networks to several variants of maximum matching problems.

Keywords: Traffic grooming, optical networks, approximation algorithms, maximum matching, Add-Drop Multiplexer.

1 Introduction

1.1 Optical Networks

All-optical networks have been largely investigated in recent years due to the promise of data transmission rates several orders of magnitudes higher than current networks [3,5]. Major applications are in video conferencing, scientific visualization and real-time medical imaging, high-speed supercomputing and distributed computing. The key to high speeds in all-optical networks is to maintain

* This research was partially supported by the Israel Science Foundation, grant No. 1249/08 and British Council grant UKTELHAI09.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 41–56, 2010.

the signal in optical form, thereby avoiding the prohibitive overhead of conversion to and from the electrical form at the intermediate nodes. The high bandwidth of the optical fiber is utilized through *Wavelength-Division Multiplexing (WDM)*: two signals connecting different source-destination pairs may share a link, provided they are transmitted on carriers having different wavelengths (or colors) of light. The optical spectrum being a scarce resource, given communication patterns in different topologies are often designed so as to minimize the total number of used colors, also as a comparison with the trivial lower bound provided by maximum load, that is the maximum number of requests sharing a same physical edge (see [10] for a survey of the main related results).

Though a lot of works have been done in the path and ring topologies, much less is known for more complex networks. The main problem in such networks is the existence of nodes of degree more than two, where traffic can split. Tree networks are thus the first candidates to be studied, and understanding the complexity of the problem for a star network is therefore central step in this direction of finding a general solution for tree networks, and then to general topology networks.

We study the grooming problem (see Section 1.2) in star networks. Despite its simplicity, this topology is important, as it arises in the interconnection of LANs (local area networks) or MANs (metropolitan area networks) with a wide area backbone.

1.2 The Problem

The focus of current studies is to consider the hardware cost. This is modeled by considering the basic switching unit of Add-Drop-Multiplexer (ADM), and focusing on the total number of these ADMs. The key point here is that each lighpath uses two ADMs, one at each endpoint. If two incident lightpaths are assigned the same wavelength, then they can share the ADM at their common endpoint. An ADM may be shared by at most two lightpaths. Moreover, in studying the hardware cost, the issue of *traffic grooming* is central. This problem stems from the fact that the network usually supports traffic that is at rates which are lower than the full wavelength capacity, and therefore the network operator has to be able to put together (= groom) low-capacity requests into the high capacity fibers. At most g requests can share one lightpath. g is termed the *grooming factor*. In terms of ADMs, if g requests of the same wavelength entering through the same edge to one node, they can all use the same ADM at that node, thus saving $g-1$ ADMs. The goal is to find a coloring that minimizes the total number of ADMs. There are two sub-models, depending on whether or not paths that have the same color can use more than two edges touching any node (*bifurcation allowed* and *bifurcation not allowed*, resp.).

In Fig. 1 a star network is presented, with three requests a, b and c. If $g = 2$ then all these requests can get the same color if traffic bifurcation is allowed, and in this case we use a total of 3 ADMs; otherwise, three colors are needed, because any set of two paths will imply a set of three edges touching the center

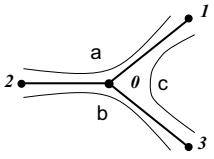


Fig. 1. Star network. Requests a, b, and c can be groomed for $g = 2$ only if traffic bifurcation is allowed.

of the star, and in this case we use a total of 6 ADMs. In this work we study the case where bifurcation is not allowed.

In graph-theoretic terms, the network is modeled as a graph, each request is viewed as a simple path in the graph, and the traffic grooming problem is viewed as assigning colors to the paths, such that, for any color λ, at most g of the paths colored λ can share the same edge.

For tree networks, and when bifurcation is not allowed, the problem reduces to partitioning the set of requests (paths) into sets (subgraphs) such that in each subgraph (a) the number of paths using any edge is at most g, and (b) the graph induced by the edges of its paths has maximum degree 2. The cost associated with such a subgraph is the number of distinct endpoints of its paths, and the goal is to find a partition that minimizes the sum of the costs of the subgraphs.

1.3 Related Works

A review on traffic grooming problems can be found in [16]. In [8] the traffic grooming problem in tree and star networks is studied. The authors extend the approximation results for ring networks of [9] to trees and stars, obtaining an approximation algorithm with approximation factor $2 \log g + o(\log g)$, running in polynomial time if g is constant. The traffic grooming model studied in [8] allows bifurcation, and in this case the problem is shown to be NP-complete even for star networks, if $g \geq 3$.

In [11] traffic grooming on path, star, and tree networks is addressed with a deep technological background, but the cost function used to minimize the cost of electronic equipment differs from that used in [2,4,9,1,13,12,8] among many others. Indeed, in [11] the authors consider the model first stated in [7] (in particular, the function referred as *total amount of electronic switching*), where one unity of cost (namely, a SONET Add-Drop Multiplexer, ADM for short) is incurred each time a wavelength conversion of a request is carried out. That is, what is intended to minimize in this model is the total number of optical hops of all requests from their origins to their destinations. It is assumed that each request requires some electronics at its origin and its destination nodes, regardless of how many requests are terminated at those nodes on each wavelength.

1.4 Summary of Results

In this work we deal with the *single hop* problem, where a request is carried along one wavelength, and where bifurcation is not allowed. We study a star network,

with $n+1$ nodes $0, 1, 2, \cdots, n$, and a set of edges $\{\{0,1\}, \{0,2\}, \cdots, \{0,n\}\}$. The requests are of length 1 or 2, termed *short* and *long* requests, respectively. We first consider the case of simple requests (Section 3), that is, there are no two requests sharing both endpoints (i.e. identical). In this case only the case $g = 2$ is of interest, and we show a polynomial-time algorithm for the problem. Actually, this result holds also for the case when multiple identical requests of length 2 are allowed. It turns out that a basic component of each solution is a triangle; that is, three requests – one between i and 0, one between j and 0, and one between i and j – all colored with the same color. Indeed, we show that in each optimal solution there is the same number of triangles. We also present (Section 4) results for the case of multiple requests and $g = 2$ (though the exact complexity of this case remains open). We present two techniques, which lead to two $\frac{4}{3}$-approximation algorithms. Our algorithms reduce our problems to the maximum matching problem and several of its variants. We start with preliminaries in Section 2, and conclude with a summary in Section 5.

2 Preliminaries

In this section we provide some preliminaries concerning matchings and the notation we use to denote the subgraphs involved in a partition of the request graph. We use standard graph terminology (cf. for instance [6]). We consider undirected graphs, which may have multiple edges and self-loops.

Matchings. Let $G = (V, E)$ be a (multi)graph with a weight function $w : E \to \mathbb{R}$, and let $\mathbf{I}$ be a function associating an interval of natural numbers with each vertex in V. An $\mathbf{I}$-*matching* is a function $\mathbf{m} : E \to \mathbb{N}$ such that for $v \in V$, $\sum_{e \in E | v \in e} \mathbf{m}(e)$ lies in the interval $\mathbf{I}(v)$. An $\mathbf{I}$-*factor* is an $\mathbf{I}$-matching such that $\mathbf{m} : E \to \{0, 1\}$. A *matching* is an $\mathbf{I}$-factor such that $\mathbf{I}(v) = [0, 1]$ for each $v \in V$. A *maximum* $\mathbf{I}$-matching is an $\mathbf{I}$-matching $\mathbf{m}$ such that $\sum_{e \in E} \mathbf{m}(e) \cdot w(e)$ is maximized. Maximum $\mathbf{I}$-matchings can be found in polynomial time [14,15]. An $\mathbf{I}$-matching $\mathbf{m}$ corresponds to a sub(multi)graph M of G, such that the multiplicity of the edges of G in M is given by the function $\mathbf{m}$. With slight abuse of notation, M will be also called an $\mathbf{I}$-matching.

Notation. The request graph is denoted R, where a request between two vertices i and j, denoted (i, j), corresponds to an edge of R. The grooming factor is denoted g. A subgraph of R is called *valid* if (a) it respects the grooming constraint, that is, no more than g requests share the same link, and (b) it contains at most two leaves of the star (this is equivalent to forbidding bifurcation). Therefore, the problem we consider can be stated as finding a minimum cost partition of $E(G)$ into valid subgraph, where the cost of a partition is given by the total number of vertices of the subgraphs involved in the partition. When the physical network is a star and the grooming factor is at most 2, the possible subgraphs involved in a partition of $E(R)$ are denoted as follows:

- i denotes the request $(i, 0)$.
- $i + j$ denotes the path on three vertices made of the two requests $(i, 0)$ and $(j, 0)$.
- $i + (i, j)$ denotes the subgraph made of the long request (i, j) and the short request $(i, 0)$.
- $[i, j]$ denotes the *triangle* that consists of the three requests $(i, 0)$, $(j, 0)$, and (i, j).
- $2(i, j)$ (resp. $2i$) denotes the subgraph made of two copies of the request (i, j) (resp. $(i, 0)$).
- $2i + j$ denotes the subgraph made of two copies of the request $(i, 0)$ and one copy of the request $(j, 0)$.
- $2i + 2j$ denotes the subgraph made of two copies of the request $(i, 0)$ and two copies of the request $(j, 0)$.

Note that last four subgraphs corresponding to the three last items are only possible if multiple requests are allowed.

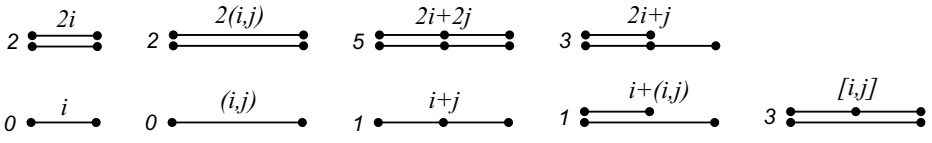


Fig. 2. Possible subgraphs in the star for $g = 2$, together with the notation used to denote them. The number near each subgraph indicates its *savings*. Note that the upper subgraphs are only possible if multiple requests are allowed.

The above subgraphs are shown in Fig. 2, together with the saving in the number of ADMs corresponding to each subgraph. For example, the case $2i + j$ corresponds to a subgraph containing two short requests $(i, 0)$ and one short request $(j, 0)$. The saving of this subgraph is 3 because if every request stood by its own, they would use a total of 6 ADMS, while by grouping them in a subgraph of type $2i + j$ they use only 3 ADMs, so the saving is of $6 - 3 = 3$ ADMs. If only simple requests are given, then we only have the subgraphs depicted in the lower row of Fig. 2, while multiple requests allow also for the subgraphs depicted in the upper row. Note that minimizing the total cost is equivalent to maximizing the total number of savings.

3 Simple Requests

We first provide an optimal algorithm in Section 3.1 and then we give a combinatorial characterization of the optimal solutions in Section 3.2.

3.1 Optimal Algorithm

Theorem 1. *For any $g \geq 1$, there exists a polynomial time exact algorithm for* Star Traffic Grooming *problem for the case of simple requests.*

Proof. Let first $g = 1$. Each long request (i, j) uses 2 ADMs, and no saving is possible. Therefore, only short requests of type i can be matched and save 1 ADM at the central node 0, so if the instance contains x short requests, $\lfloor \frac{x}{2} \rfloor$ savings can be done, and such a solution is optimal.

If grooming is allowed, but no multiple requests are allowed, then note that the case $g = 2$ is the only interesting case, since for arbitrary $g \geq 2$, as we do not allow bifurcations, the only possible subgraphs are those depicted in the lower row of Fig. 2. So, we focus henceforth on the case $g = 2$.

We claim that the following Algorithm SimpleMatch

Algorithm SimpleMatch:
Input: A star S and a set R of *simple* requests between pairs of vertices in S.
Output: a partition of $E(R)$ into a set of valid subgraphs for $g = 2$.

(1) Build an edge-weighted graph $G = (V, E)$ as follows:
  (1.1) For each short request $(i, 0)$ in R, add a vertex v_i to V.
  (1.2) For each long request (i, j) in R, add a vertex v_{ij} to V.
  (1.3) For each vertex $v_{ij} \in V$, if the vertex v_i (resp. v_j) is in V, add the edge $\{v_{ij}, v_i\}$ (resp. $\{v_{ij}, v_j\}$) to E with weight 1.
  (1.4) For each pair of vertices $v_i, v_j \in V$, add the edge $\{v_i, v_j\}$ to E with weight 3 if the vertex v_{ij} is in V, and add it with weight 1 otherwise.
(2) Find a maximum weighted matching M in G.
(3) Output the set of subgraphs corresponding to the edges in M. If some request is left, output it as a subgraph itself.

returns an optimal solution in polynomial time.

The possible subgraphs involved in the partition of the request set are the five lower subgraphs of Fig. 2. We only need to show that all the possible subgraphs, together with their savings, correspond to the weighted edges of the graph G build in the description of the algorithm.

The subgraphs of type $i + (i, j)$ are captured by the edges $\{v_{ij}, v_i\}$, and their weight does correspond to the unitary savings. Also the subgraphs of type $i + j$ when the request $(i, j) \notin R$ are captured by the edge $\{v_i, v_j\}$ with weight 1. Finally, let us see how triangles are represented in G. The key idea is the following.

Claim 2. *If for two integers i, j the three requests $(i, 0)$, $(j, 0)$, and (i, j) are in R, there exists an optimal solution not containing the subgraph $i + j$.*

Indeed, assume that an optimal solution with cost OPT contains the subgraph $i + j$. We can remove the request (i, j) from wherever it is, add it to the subgraph $i + j$, and then the obtained solution containing the triangle $[i, j]$ has cost $OPT' \leq OPT$.

Due to Claim 2, whenever three requests $(i,0)$, $(j,0)$, and (i,j) belong to R, we do not need to consider the subgraph $i+j$. In this case, the edge $\{v_i, v_j\}$ with weight 3 corresponds to the triangle $[i,j]$. Note that if the edge $\{v_i, v_j\}$ is in the matching M, it prevents the vertex v_{ij} to be used again, since the only neighbors of v_{ij} in G are v_i and v_j.

Thus, a maximum weighted matching in G (plus possibly, some single requests) corresponds to a valid solution for the instance R maximizing the total savings, or equivalently minimizing the total number of ADMs.

In fact, when the requests are simple, for any $g \geq 3$ the problem is also solved by Algorithm SimpleMatch. This is because, as we mentioned above, due to the model that we are assuming the set of subgraphs involved in any partition of the set of requests is the same for any $g \geq 2$. □

3.2 Structure of an Optimal Solution

Assume that the maximal number of triangles in a given instance of the problem is k. We show now that *all* optimal solutions contain k triangles. Note that even if Algorithm SimpleMatch does not take into account this property, Lemma 1 provides structural information about the optimal solutions, which is of interest by itself.

We are given a set of requests $R = \{\ell_1, \cdots, \ell_x, s_1, \cdots, s_y\}$, where $\ell_1, \cdots, \ell_x$ are requests of length 2 and $s_1, \cdots, s_y$ are requests of length 1. The maximal number of triangles in R is denoted by $\max(R)$.

Lemma 1. *Let R be a given set of requests on a star network, and let $\max(R) = k \geq 0$, let $MAX = \{t_1 = [1,2], t_2 = [3,4], \cdots, t_k = [2k-1, 2k]\}$ be a maximal set of triangles in R, and let OPT be any optimal solution for R. Then OPT contains exactly k triangles.*

Proof. Assume that OPT does not contain exactly m triangles of MAX where $0 < m \leq k$ (so, OPT contains $k-m$ of the triangles in MAX) . W.l.o.g. assume these are the triangles $t_1, t_2, \cdots, t_m$. We define a function f that will assign to each triangle in $t_1, t_2, \cdots, t_m$ a triangle or a pair of triangles in $OPT - MAX$.

Consider any of these triangles $t_j = [2j-1, 2j]$, $1 \leq j \leq m$.

Since $t_j \notin OPT$, we have to consider two cases, according to whether the request $(2j-1, 2j)$ is paired in OPT with one short request (Case a) or it is by itself in OPT (Case b):

Case a: $(2j-1, 2j)$ is paired in OPT with $(2j-1, 0)$ (or, similarly, with $(2j, 0)$). Consider the request $(2j, 0)$. If $(2j, 0)$ is paired with one short request, or with one long request, or it is by itself in OPT, then by moving $(2j, 0)$ to join $(2j-1, 2j)$ and $(2j-1, 0)$ we get a solution SOL with $cost(SOL) < cost(OPT)$, a contradiction. So we conclude that in this case $(2j, 0)$ is in another triangle t (which is clearly neither of the triangles $t_1, t_2, \cdots, t_m$). In this case we define $f(t_j) = t$.

Case b: $(2j-1, 2j)$ is a component containing only itself in OPT.

Consider the request $(2j-1, 0)$ and $(2j, 0)$. They cannot form together one component of OPT, they cannot form two components with two other short requests, and neither of them can be a component of itself in OPT; this is since in each of these cases we could add them to $(2j-1, 2j)$ and get a solution SOL with $cost(SOL) < cost(OPT)$, a contradiction. Hence, they are matched to long requests in OPT. They cannot both be paired to long requests alone, since then a solution SOL that will add them to $(2j-1, 2j)$ will satisfy $cost(SOL) < cost(OPT)$, a contradiction. So, at least one of them is in a triangle t of OPT. In this case we define $f(t_j) = t$. If both of them are connected to triangles t_j^1 and t_j^2 then we define $f(t_j) = \{t_j^1, t_j^2\}$.

At this point, the set of triangles in $t_1, t_2, \cdots, t_m$ is partitioned into two subsets T_1 and T_2 of sizes s_1 and S_2, respectively, where $s_1 + s_2 = m$. Each triangle in T_1 is mapped to one triangle in OPT. We claim that this mapping is 1-1. Otherwise there are two triangles t_j and $t_{j'}$ mapped to the same triangle of OPT. This is because w.l.o.g. OPT contains the components $2j-1+(2j-1, 2j)$ and $2j'-1+(2j'-1, 2j')$ and the triangle $[2j, 2j']$; these components contribute 9 ADMs to the solution. The solution OPT' obtained from OPT by taking out all these components and adding $t_j, t_{j'}$ and $(2j, 2j')$ uses one less ADMs, a contradiction. These triangles correspond to s_1 triangles of OPT.

Each triangle in T_2 is mapped to two triangles in OPT. These triangles in OPT contain the short requests $(2j-1, 0)$ and $(2j, 0)$. The number of vertices in these triangles is thus at least $s_2 > 0$, and therefore that $\cup_{t \in T_2} f(t)$ contain at least s_2 triangles. Since it must contain at most s_2 triangles, it follows that the number of these triangles is exactly s_2, and thus the total number of triangles in OPT in $\cup_{t \in T_1 \cup T_2} f(t)$ is m. These, together with the $k-m$ triangles in OPT that belong to MAX, sum up to a total of k triangles in OPT. This completes the proof. □

4 Multiple Requests

4.1 Motivation

If multiple requests are allowed, the problem becomes more complicated. Besides the subgraphs of type $2(i, j)$, the greedy removal of any type of subgraph will not lead to an optimal solution. First we show an example where the pairing of identical short requests leads to a sub-optimal solution: Consider a star with 3 leaves a, b, and c with the request set $R = \{a, a, b, c, (a, b), (a, c)\}$. If we pair the two short requests a in the same component, the cost will be at least 8 whereas an optimal solution uses two triangles $[a, b]$ and $[a, c]$. One could be tempted to remove greedily all the triangles, or similarly a set of triangles of maximum cardinality. The following is a counter example showing that this strategy is not optimal. Consider the request set $R = \{a, a, b, b, (a, b)\}$. A solution containing the only triangle $[a, b]$ will have a cost of at least 6 ADMs, as opposed to the cost of 5 ADMs in the optimal solution $(a, b), 2a + 2b$.

First we provide in Subsection 4.2 an approximation algorithm in the same spirit of Section 3. Namely, we construct an auxiliary edge-weighted graph G made of appropriate gadgets that capture the possible subgraphs involved in

a partition of the request graph, and then we find a feasible solution to our problem by optimally solving a maximum **I**-matching problem in G. We then present another approach in Subsection 4.3, also based on **I**-matching. We prove that these two apparently different algorithms constitute a 4/3-approximation to the problem for $g = 2$. Before presenting the algorithms, we make an observation to be used in the sequel.

Claim 3. *The subgraphs of type* $2(i, j)$ *can be greedily removed from* R *without changing the cost of an optimal solution.*

Proof. Assume that R contains two copies of the request (i, j), and that in an optimal solution OPT these copies belong to different subgraphs B_1 and B_2. Since $|V(B_1)| \geq 2$ and $|V(B_2)| \geq 2$, the solution OPT' obtained from OPT by replacing the subgraphs B_1 and B_2 with $B'_1 = (B_1 \setminus (i, j)) \cup (B_2 \setminus (i, j))$ and $B'_2 = 2(i, j)$ satisfies $cost(OPT') \leq cost(OPT)$. □

By Claim 3, we assume henceforth that the requests of type (i, j) (i.e., long requests) are simple. For any leaf i of a star S, denote by s_i (resp. ℓ_i) the number of *short* (resp. *long*) requests of node i in the request set R. We also let $SH = \sum_i s_i$ and $L = \sum_i \ell_i$.

Given a solution of the problem, we say that a request i *paired* if its subgraph in this solution contains another request i, i.e., i is in a subgraph of type $2i+2j$, $2i + j$, or $2i$, and *unpaired* otherwise.

4.2 First Approach

Our first approach is described in Algorithm MultipleMatch1.

Algorithm MultipleMatch1:
Input: A star S and a set R of (possibly *multiple*) requests between pairs of vertices in S.
Output: a partition of $E(R)$ into a set of valid subgraphs for $g = 2$.

(1) Build an auxiliary edge-weighted multigraph $G = (V, E)$ as follows:
  (1.1) For each leaf i in S, add a new vertex v_i to V.
  (1.2) For each vertex $v_i \in V$, add a self-loop in v_i to E with weight 2.
  (1.3) For each long request (i, j) in R, add two new vertices v_{ij}, v'_{ij} to V and the following edges to E: $\{v_i, v_{ij}\}$ with weight 1, $\{v_j, v'_{ij}\}$ with weight 1, $\{v_{ij}, v'_{ij}\}$ with weight 1, and $\{v_{ij}, v'_{ij}\}$ with weight -1.
(2) Define the following function **I**, which associates an interval of natural numbers with each vertex in V:
  (2.1) $\mathbf{I}(v_i) = [0, s_i]$ for each $v_i \in V$.
  (2.2) $\mathbf{I}(v_{ij}) = \mathbf{I}(v'_{ij}) = [2, 3]$ for each $v_{ij}, v'_{ij} \in V$.
(3) Find a maximum **I**-matching M of G using the algorithm of [14].
(4) Output the following subgraphs of R according to M:
  (4.1) For each self-loop of vertex v_i in M, output subgraph $2i$.
  (4.2) For each i, j in S, if the two edges $\{v_i, v_{ij}\}, \{v_j, v'_{ij}\}$ and the edge $\{v_{ij}, v'_{ij}\}$ with weight 1 are in M, output subgraph $[i, j]$.
  (4.3) For each i, j in S, if the edge $\{v_i, v_{ij}\}$ and the two copies of the edge $\{v_{ij}, v'_{ij}\}$ are in M, output subgraph $i + (i, j)$.
  (4.4) For each i, j in S, if the edge $\{v_j, v'_{ij}\}$ and the two copies of the edge $\{v_{ij}, v'_{ij}\}$ are in M, output subgraph $j + (i, j)$.
  (4.5) If some request is left, output it as a subgraph itself.

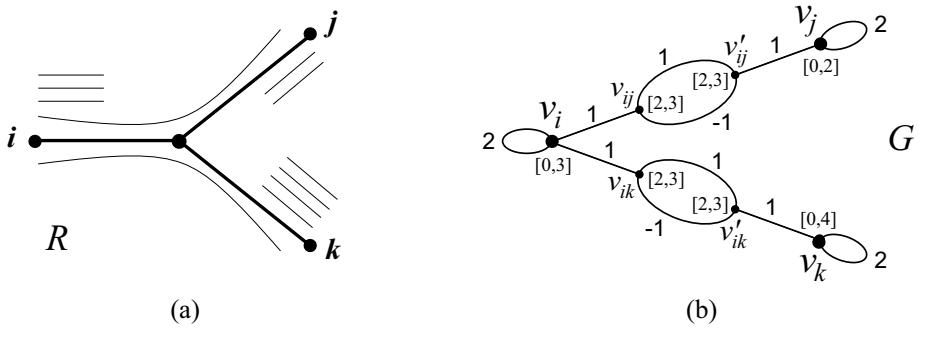


Fig. 3. (a) A traffic instance I in a star on three vertices i, j, k, with $s_i = 3$, $s_j = 2$, and $s_k = 4$. (b) Auxiliary graph G and function $\mathbf{I}$ built in Algorithm MultipleMatch1. The number beside each edge indicates its weight, while the interval in brackets beside each vertex indicates its allowed degrees.

An example of the graph G and the function $\mathbf{I}$ built in Algorithm MultipleMatch1 is illustrated in Fig. 3 for a simple star on four vertices. We now briefly discuss the intuition behind the algorithm. For each leaf i of S, the degree bounds at node v_i assure that no more than s_i short requests can be used at vertex i. The degree bounds at vertices v_{ij}, v'_{ij} make sure that the output of the $\mathbf{I}$-matching algorithm can be indeed translated to a partition of the requests in R, and such the savings of each subgraph correspond to the sum of the weights of the edges in M corresponding to this subgraph. As one can check from step (4) of Algorithm MultipleMatch1, we do not use any subgraph of type $2i + 2j$, $2i + j$, or $i + j$. The key point is that the gadgets used by the algorithm capture simultaneously all other possible subgraphs. The fact of *forgetting* some subgraphs has of course direct consequences in the worst-case performance of the algorithm, as stated in the following theorem.

Theorem 4. *Algorithm* MultipleMatch1 *is a polynomial-time* 4/3*-approximation algorithm for the* Star Traffic Grooming *problem for* $g = 2$ *when multiple requests are allowed.*

Proof. First, the algorithm clearly runs in polynomial time, as the algorithm of [14] used in step (3) does so. We now argue that the output of the algorithm defines a feasible solution to Star Traffic Grooming for $g = 2$. From steps (4.1)-(4.5) it follows that the number of short requests used at each vertex i of S by the output of Algorithm MultipleMatch1 is equal to the degree (taking into account the multiplicity of the edges, and considering that a self-loop induces degree 2) of v_i in the $\mathbf{I}$-matching M of G. Since by definition of $\mathbf{I}(v)$, the degree of v_i in M is at most s_i, no more than s_i short requests are used at vertex i.

In order to analyze the approximation ratio of the algorithm, we now discuss how the output M reflects the cost of the solution of Star Traffic Grooming that it defines. (Recall Fig. 2 for the definition of the possible subgraphs and their associated savings.) In case (4.1), for each vertex i in S, a subgraph of type $2i$ has an associated saving of 2, which corresponds to the weight of a self-loop at vertex v_i in G. Let i, j be two vertices in S such that the long request (i, j) is in R. In case (4.2), if the two edges $\{v_i, v_{ij}\}, \{v_j, v'_{ij}\}$ and one copy of $\{v_{ij}, v'_{ij}\}$ belong to M, then the fact that M is a maximum **I**-matching implies that the copy of $\{v_{ij}, v'_{ij}\}$ in M is the one with positive weight. In this case, the sum of the weights of these three edges is 3, which is equal to the saving of a triangle $[i, j]$. In case (4.3), the edge $\{v_i, v_{ij}\}$ and the two copies of the edge $\{v_{ij}, v'_{ij}\}$ are in M, so the sum of the weights of these three edges is 1, which corresponds to the saving of the subgraph $i + (i, j)$. Case (4.4) is symmetric to case (4.3). If none of cases (4.2), (4.3), or (4.4) holds, then none of the edges $\{v_i, v_{ij}\}$ and $\{v_j, v'_{ij}\}$ is in M. From the degree constraints at vertices v_{ij}, v'_{ij}, one can check that the only remaining feasible possibility for an **I**-matching is to take both copies of the edge $\{v_{ij}, v'_{ij}\}$, therefore incurring a total weight of 0, which indeed corresponds to not saving any ADM.

Finally, in case (4.5), a subgraph of R made of a single request has a saving of 0.

From the above discussion, it follows that there is a bijective correspondence between the gadgets used by the **I**-matching and subgraphs of type $2i$, $[i, j]$, or $i + (i, j)$. Therefore, our algorithm finds the best solution *under the constraint* of not using the other subgraphs that incur some saving, namely those of type $2i + 2j$, $2i + j$, or $i + j$ (see Fig. 2).

Given an instance R, let $cost(OPT)$ be the cost of an optimal solution OPT, and let $cost(ALG)$ be the cost of a solution ALG given by Algorithm MultipleMatch1. We construct from OPT another solution SOL as follows. For every subgraph B of type $2i + 2j$, $2i + j$, or $i + j$ in OPT, we split B into two subgraphs B_i and B_j, where B_i (resp. B_j) contains the short requests of vertex i (resp. j). Note that for each such subgraph B, the cost of B in OPT is 3, while the cost of B_i plus the cost of B_j in SOL is 4. As all the other subgraphs remain unchanged, it follows that $cost(SOL) \leq \frac{4}{3} \cdot cost(OPT)$. But as no subgraph of type $2i+2j$, $2i+j$, or $i+j$ is in solution SOL, the solution found by Algorithm MultipleMatch1 is equal or better than SOL, so $cost(ALG) \leq cost(SOL)$, which in turn implies that $cost(ALG) \leq \frac{4}{3} \cdot cost(OPT)$. □

4.3 Second Approach

In this section we propose an algorithm that uses an oracle (called TriangleOracle in the algorithm) providing part of the subgraphs, namely all the triangles $[i, j]$ of the solution. We show that the algorithm is optimal provided that the oracle is optimal. In this way we reduce the problem to the problem of finding an optimal set of triangles.

Algorithm MULTIPLEMATCH2:
Input: A star S and a set R of (possibly *multiple*) requests between pairs of vertices in S.
Output: a partition of $E(R)$ into a set of valid subgraphs for $g = 2$.

(1) Build an unweighted graph $G = (V, E)$ as follows:
  (1.1) V is the set of leaves of S, that is $V = \{1, \ldots, n\}$.
  (1.2) $\{i, j\} \in E$ whenever the request (i, j) belongs to R.
// *Note that* $d_G(i) = \ell_i$ *for each* $i = 1, \ldots, n$.
(2) Invoke the algorithm TRIANGLEORACLE$(G, \mathbf{s})$, where $\mathbf{s}$ is the vector of the values s_i of the number of short requests i. The algorithm returns a subgraph T of G.
(3) For each edge $\{i, j\} \in E(T)$, return the triangle $[i, j]$ as a subgraph and remove it from R.
(4) For each node $i \in V(G)$ build $\lfloor \frac{s_i - d_T(i)}{2} \rfloor$ subgraphs of type $2i$ and at most one subgraph of type i (but do not remove them from R).
(5) For each subgraph of type i built in the previous step, choose arbitrarily a request $(i, j) \in R$, build the subgraph $i + (i, j)$ and remove it from R. If no such request exists, do nothing.
(6) Build a complete n-partite graph whose nodes are the subgraphs built at step (4) that are still in R. There is an edge between two subgraphs if they correspond to different nodes of S. Calculate a maximum matching of this graph. Each edge of this matching corresponds to a subgraph of the form $2i + 2j$, $2i + j$ or $i + j$, and each unmatched node of corresponds to a subgraph of type i. Return all these subgraphs.

From the above description of the algorithm, it follows that the set of subgraphs of type $[i, j]$ (i.e. triangles) returned by algorithm MULTIPLEMATCH2 corresponds to the edges $E(T)$ of the subgraph returned by TRIANGLEORACLE. For this reason, in the rest of this section we will use the terms *edge* and *triangle* interchangeably. In order algorithm MULTIPLEMATCH2 to be optimal, it is a necessary condition that TRIANGLEORACLE returns the set of triangles of an optimal solution. We will prove that this is also a sufficient condition.

Theorem 5. *If the set of triangles $E(T)$ returned by* TRIANGLEORACLE *is the set of triangles of some optimal solution, then* MULTIPLEMATCH2 *returns an optimal solution.*

Proof. We have to show that the decisions taken at steps (4), (5) and (6) are correct. We start with step (4).

We claim that there is an optimal solution that does not contain two subgraphs G_1 and G_2 such that both contain an unpaired request i. This implies that the decision taken by the algorithm in step (4) is optimal.

Indeed, consider an optimal solution with the same set of triangles returned by our algorithm, i.e. the set $E(T)$. Assume by contradiction that it contains two such subgraphs G_1 and G_2. If one of these subgraphs (without loss of generality G_1) does not contain a long request (i, j), then the request i in G_2 can be moved to the subgraph G_1. In this case it does not increase the cost of G_1, because it shares the ADMs of the request i in G_1, and its removal from G_2 does not increase the cost of G_2. Otherwise both G_1 and G_2 contain long requests (i, j_1) and (i, j_2) respectively. Moreover, $j_1 \neq j_2$ because by Claim 3 we assume that the long requests form a simple set of requests. Also neither one of G_1 and G_2 is a triangle, because these triangles are not returned by our algorithm, and therefore are not in this optimal solution. Therefore $G_1 = i + (i, j_1)$ and $G_2 = i + (i, j_2)$ and they use 6 ADMs in total. In this case we can replace G_1 and G_2 with the

three subgraphs $2i$, (i, j_1), and (i, j_2) having a total cost of 6 ADMs, to obtain an optimal solution with the claimed property.

We proceed with the correctness of step (5). In the rest of the proof we consider the cost of a solution in the following way. The L long requests incur a fixed cost of $2L$ ADMs. All other (i.e. short) requests sharing one of these ADMs do not incur any cost for these ADMs. The first observation is that, if there is an unpaired request i at the beginning of this step, a subgraph of type $i + (i, j)$ is an optimal subgraph for it. This is because it incurs a cost of at most 1 ADM (at node 0) in this case, and in any other case it will incur a cost of at least 1 ADM (at node i). The second observation is that no two unpaired requests i and j and a long request (i, j) can exist in R at this point. This is because in this case they would incur a total cost of 1 in the triangle $[i, j]$, and in any other subgraph each one of them incurs a cost of 1, contradicting the optimality of the triangles returned by TriangleOracle. In other words, no two short requests i and j can "compete" for a long request (i, j), therefore greedily forming the subgraphs $i + (i, j)$ will not cause a conflict.

The correctness of step (6) is now almost straightforward. At this point we are left with subgraphs of type i and $2i$ which might be merged to form bigger subgraphs. It can be checked that all the other subgraphs can not be merged. Moreover, these subgraphs can be merged only in pairs, namely pairs of the form $i + j$, $2i + j$, or $2i + 2j$. Each such merging operation reduces the cost of the solution by one ADM. Therefore the goal of the algorithm is to maximize the number of these merging operations. This is equivalent to calculate the maximum cardinality matching of the auxiliary graph built in step (6). □

Following the above result, our goal is to find an algorithm TriangleOracle(G, s) that returns the set of triangles of some optimal solution. Having in mind the counter examples presented at the beginning of this section, we present the following lemma that gives a partial characterization of an optimal set of triangles.

Lemma 2. *There is an optimal solution OPT with the following property: Let T be the set of triangles of OPT and let $[i, j] \in T$. Then either $s_i - d_T(i)$ is even or OPT contains a subgraph $i + (i, j)$.*

Proof. We consider the optimal solution OPT that having as set of triangles the set T returned by the oracle. Let us also consider a triangle $[i, j] \in T$. Assume that $s_i - d_T(i)$ is odd and there is no subgraph $i + (i, j)$ in OPT. Then after step (3) there will be an odd number of i requests left in R. Therefore there will be one unpaired request i after step (4). This request will not participate in a subgraph $i + (i, j)$ at step (5) by the assumption. Therefore it will remain unpaired in OPT. In this case we can obtain a solution OPT' from OPT by removing the request i from the triangle $[i, j]$, and moving it to the subgraph containing this unpaired request without increasing the cost. If j has also this property then this leads to a contradiction to the optimality of OPT, otherwise OPT' is the claimed optimal solution. □

By Lemma 2 we can restrict ourselves to algorithms not returning any triangle $[i, j]$ in T if this causes $s_i - d_T(i)$ (or $s_j - d_T(j)$) to be odd. For this reason we propose the following algorithm as a first attempt towards a TriangleOracle.

Algorithm TrianglesViaIFactor:
Input: A Graph $G = (V, E)$, and a vector $\mathbf{s}$ of numbers indexed by V.
Output: A subgraph T of G.

(1) Define the function $\mathbf{f} : V \to \mathbb{N}$ as follows:

$$\mathbf{f}(i) = \begin{cases} s_i, & \text{if } s_i \leq \ell_i \\ \ell_i - (s_i - \ell_i) \bmod 2, & \text{otherwise} \end{cases}$$

(2) Find a maximum $\mathbf{I}$-factor T in G, where $\mathbf{I}(i) = [0, \mathbf{f}(i)]$ for each $i \in V$.

Theorem 6. MultipleMatch2 *is a* 4/3*-approximation for the* Star Traffic Grooming *problem if it uses* TrianglesViaIFactor *as* TriangleOracle.

Proof. Let ALG be a solution returned by the algorithm and let OPT be an optimal solution. Let T' be the set of triangles of $ALG \setminus OPT$. Consider the $2|T'|$ short requests in these triangles (of ALG). Let x be the number of short requests participating in the same subgraphs as these requests in OPT. Then the cost of OPT satisfies

$$cost(OPT) \;\geq\; \frac{3}{4}(2|T'| + x) = \frac{3}{2}|T'| + \frac{3}{4}x$$

because these subgraphs are not triangles by the way T' is chosen, and in any subgraph of another type a short request incurs a cost of 3/4 in average, the best case being a subgraph of type $2i + 2j$ using 3 ADMs for 3 short requests.

On the other hand, the cost of ALG satisfies

$$cost(ALG) \;\leq\; |T'| + x \leq \frac{3}{2}|T'| + x$$

because all these x requests are either paired in S, or part of a subgraph of type $i + (i, j)$. In both cases each such request incurs a cost of at most 1. Comparing the right hand sides of the above inequalities we conclude the claim. □

However the proposed algorithm is not optimal, as the following lemma shows:

Lemma 3. *There is an instance for which* MultipleMatch2 *using* TrianglesViaIFactor *as* TriangleOracle *returns a sub-optimal solution.*

Proof. Consider the following instance, on a star with 4 leaves, with $R = \{(1, 2), (2, 3), (3, 4), (4, 1), 1, 2, 2, 2, 2, 4, 4, 4, 4, 4\}$.

In this case G is a C_4, $\ell = (1, 1, 1, 1)$, $\mathbf{s} = (1, 4, 0, 5)$ the $\mathbf{f} = (1, 2, 0, 1)$, and any maximum $\mathbf{I}$-factor has cardinality 1. The oracle might return the singleton $T = \{(1, 2)\}$ which leads to a solution with a cost of 16 ADMs. On the other hand, there is a solution with $T^* = \{(4, 1)\}$ as the set of its triangles, implying a cost 15 ADMs. □

5 Conclusions

We studied the traffic grooming problem in star networks when bifurcation is not allowed. We presented a polynomial-time algorithm for the case of simple requests, and gave some insight into the structure of an optimal solution. Though the algorithm can be extended to the case of multiple long requests, the complexity of the problem when multiple short requests are allowed remains unsolved. We presented two approaches with good approximation guarantee using matching techniques. We expect our techniques to lead to a polynomial-time algorithm for the case $g = 2$. For instance, in the approach we presented in Section 4.2, in order to obtain a polynomial-time algorithm for the problem it would be enough to find the right gadgets that also capture the *missing* subgraphs. In fact, the only subgraph we have not been able to capture is the *quadruple* given by two pairs of short requests, so we suspect that either such a gadget may be found, or the quadruple would probably play a distinguished role in a possible NP-completeness proof.

While, according to our results, matching techniques prove to be very helpful for the case $g = 2$, it is not clear how to use them for $g > 2$. It might well be the case that more complicated techniques are needed to deal with higher values of the grooming factor, even for this apparently simple network topology. We believe that our study sheds light on the complexity of traffic grooming for networks whose maximal degree is more than two. Among them, it will be of interest to study the complexity of the problem for tree networks, bounded degree networks, or planar networks.

References

1. Amini, O., Pérennes, S., Sau, I.: Hardness and Approximation of Traffic Grooming. Theoretical Computer Science 410(38-40), 3751–3760 (2009)
2. Bermond, J.-C., Braud, L., Coudert, D.: Traffic grooming on the path. Theoretical Computer Science 384(2-3), 139–151 (2007)
3. Brackett, C.A.: Dense wavelength division multiplexing networks: principles and applications. IEEE Journal on Selected Areas in Communications 8, 948–964 (1990)
4. Chow, T., Lin, P.: The ring grooming problem. Networks 44(3), 194–202 (2004)
5. Chung, N.K., Nosu, K., Winzer, G.: Special issue on dense wdm networks. IEEE Journal on Selected Areas in Communications 8 (1990)
6. Diestel, R.: Graph Theory, vol. 173. Springer, Heidelberg (2005)
7. Dutta, R., Rouskas, N.: Traffic grooming in WDM networks: Past and future. IEEE Network 16(6), 46–56 (2002)
8. Flammini, M., Monaco, G., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the Traffic Grooming Problem in Tree and Star Networks. Journal of Parallel and Distributed Computing 68(7), 939–948 (2008)
9. Flammini, M., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the Traffic Grooming Problem. Journal of Discrete Algorithms 6(3), 472–479 (2008)

10. Gargano, L., Vaccaro, U.: Routing in All–Optical Networks: Algorithmic and Graph-Theoretic Problems. In: Numbers, Information and Complexity, Kluwer Academic, Dordrecht (2000)
11. Huang, S., Dutta, R., Rouskas, G.: Traffic Grooming in Path, Star, and Tree Networks: Complexity, Bounds, and Algorithms. IEEE Journal on Selected Areas in Communications 24(4), 66–82 (2006)
12. Li, Z., Sau, I.: Graph Partitioning and Traffic Groomingwith Bounded Degree Request Graph. In: Paul, C. (ed.) WG 2009. LNCS, vol. 5911, pp. 250–261. Springer, Heidelberg (2009)
13. Muñoz, X., Sau, I.: Traffic Grooming in Unidirectional WDM Rings with Bounded Degree Request Graph. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 300–311. Springer, Heidelberg (2008)
14. Pulleyblank, R.: Faces of Matching Polyhedra. PhD thesis, University of Waterloo (1973)
15. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Heidelberg (2003)
16. Zhu, K., Mukherjee, B.: A review of traffic grooming in wdm optical networks: Architecture and challenges. Optical Networks Magazine 4(2), 55–64 (2003)

Event Extent Estimation*

Marcin Bienkowski[1], Leszek Gąsieniec[2], Marek Klonowski[3], Miroslaw Korzeniowski[3], and Stefan Schmid[4]

[1] Institute of Computer Science, University of Wrocław, Poland
[2] Department of Computer Science, The University of Liverpool, UK
[3] Wroclaw University of Technology, Poland
[4] Deutsche Telekom Laboratories / TU Berlin, Germany

Abstract. This paper studies local-control strategies to estimate the size of a certain event affecting an arbitrary connected subset of nodes in a network. For example, our algorithms allow nodes in a peer-to-peer system to explore the remaining connected components after a Denial-of-Service attack, or nodes in a sensor network to assess the magnitude of a certain environmental event. In our model, each node can keep some extra information about its neighborhood computed during the deployment phase of the network. On the arrival of the event, the goal of the active nodes is to learn the network topology induced by the event, without the help of the remaining nodes. This paper studies the tradeoffs between message and time complexity of possible distributed solutions.

1 Introduction

This paper attends to the problem of how nodes in a network can efficiently learn about (or deal with) the effects of a certain event. We assume that before the event takes place, e.g., during network deployment, nodes have sufficient time to perform certain pre-computations. Then, at some unknown time point, an event activates an arbitrary subset of nodes. We investigate distributed algorithms that allow the activated nodes to gather necessary information about the event in an efficient, cooperative manner — without the help of the remaining nodes. In particular, our algorithms allow these nodes to learn the topology induced by affected nodes.

For example, consider a peer-to-peer network which is hit by a virus spreading along the topology, or which is under a denial-of-service attack. After the attack, the goal of the surviving peers is to learn about the remaining functional peers in their respective connected component, e.g., in order to trigger a best-effort recovery of both data and topology.

Natural disaster detection is another motivation for our model. Today, many observation systems are used to monitor a certain endangered area and to warn about floods, fires, or earthquakes in time, to prevent larger damage. Besides

* Supported by MNiSW grant number N206 257335, 2008-2011, PBZ/MNiSW/07/2006/46.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 57–71, 2010.

global solutions like satellite systems, there is a trend towards distributed monitoring with wireless sensor nodes that are distributed in space (see e.g. [3] for detecting seismic events). Our algorithms can be useful in this context: they allow for computing the number of other nodes observing a certain phenomenon.

In our work, we assume that the nodes which are not activated by the event are inactive (or faulty) and *cannot* participate in the communication nor in the computation. This is clear in our peer-to-peer example, as the nodes attacked by the virus may simply be down or crashed. In order to motivate this assumption in a sensor network, consider the situation where nodes are in an energy-parsimonious sleeping mode and only wake up in case of some external physical influence, triggered by the event.

1.1 Model

This work assumes an arbitrary undirected network or graph $G = (V, E)$ of $n = |V|$ nodes and $m = |E|$ edges. We consider a synchronous model where algorithms proceed in rounds and where the event happens at a globally unique time t_0. At this time, a subset of the nodes become *active*, the remaining ones are *inactive*. The goal of the active nodes is to find out about other active nodes.

In our model, we assume that the nodes have sufficient time to perform arbitrary *preprocessing operations* at times $t < t_0$, e.g., to learn the topology. In particular, we assume that nodes choose unique IDs from $\{1, ..., n\}$ in the preprocessing stage. Our algorithms hence work in two stages, where in the first "offline" stage nodes do *preprocessing*, and in the second "event" stage, *the actual event is explored*. During our analysis, we are mainly interested in the complexities of the second stage. A novelty of the problem studied in this paper comes from this division.

Henceforth, by $V' \subseteq V$ we denote the subset of active nodes and by G' subgraph of G induced by V'. We note that G' is not necessarily connected; we call connected components of G' *active components*. Let $s = |V'|$ be the number of active nodes, and let δ denote the *active diameter*, i.e. the maximum diameter of an active component.[1] We aim at designing algorithms in which all active nodes will learn about their active components; in particular, nodes will learn the size of their components. Observe that usually, once there is at least one node $v_{G'} \in G'$ with this knowledge in a component G', $v_{G'}$ can subsequently inform all other nodes in G' along a corresponding spanning tree. Hence, in the following, we sometimes concentrate on this case only, given that sending this information along the spanning tree does not change the asymptotic complexity.

We strive to optimize two criteria: First, our algorithms should have a good reaction time (*time complexity*). We measure the time (i.e., the number of rounds) until all active nodes know the IDs of the other active nodes in their active component. In each round, the nodes can perform arbitrary local computations and communicate with their neighbors in the network.

[1] Observe that the active diameter can be much larger than the diameter. For instance, in a $\sqrt{n} \times \sqrt{n}$-grid, the active diameter can be linear in n.

Second, we want the algorithm to send as few messages as possible (*message complexity*). We assume that in a round, each node can send a (different) message to each neighbor. Each such message costs a unit of energy, independently of the neighbor being active or not. We are interested in the total number of messages sent by all active nodes and our focus is on feasibility, so — in the lines of the $\mathcal{LOCAL}$ model [11] — we do not restrict the size of messages. Notice however that at time t_0 nodes do not know which of their neighbors are active and which are not. They have to deduce this knowledge from the protocol and the communication pattern in subsequent rounds.

1.2 Related Work

There is a considerable scientific interest in distributed monitoring and alarming systems. For instance, there are approaches to detect the boundaries of a toxic leach [6] or monitoring mechanisms to defend against Internet worms [7]. Our work is also related to literature on robustness of overlay networks, where nodes need to reorganize after an attack in an efficient manner [5].

Our paper belongs to the field of local algorithms where computations are performed by repeated interactions of nodes with their neighbors. Our problem formulation is reminiscent of classic problems such as leader election, and indeed, most of our algorithms implicitly solve this problem. However, we can identify at least two interesting and new aspects of our model. First, our algorithms adapt themselves to the environment, in the sense that the runtime and the number of messages is smaller for fewer active nodes. Several papers recently investigated local solutions for global problems for which the runtime depends on the concrete problem input [2], rather than considering the worst-case over all possible inputs: if in a special instance of a problem the input behaves well, a solution can be computed quickly. Second, we assume that only active nodes can participate in the computations, while the number of active nodes is not known in advance. Thus, preprocessing the graph appears to be of limited use, as any precomputed coordination points or infrastructure may not be active later. This poses a higher demand on efficient coordination primitives during runtime.

It is interesting to compare our work to the disaster disclosure problem introduced by Mans et al. [9] — the closest paper to our work. In [9], it is assumed that nodes that did not sense an event can participate in the coordinated exploration of a disaster. For example, this so-called "on-duty model" is meaningful in sensor networks where all nodes are regularly "online" in order to exchange status updates and can start collaborating on demand. In contrast, in the model studied in our work, we try to capture a scenario where certain nodes are not available during the distributed computations after the event; this so-called "off-duty model" has been stated as an open research direction in [9]. Despite the similar nature of the on-duty and the off-duty model, the two problems exhibit a different structure. This is due to the fact that in the on-duty model, nodes can heavily rely on computations done during the pre-processing phase. Indeed, the approach taken in [9] relies on a hierarchy of "leaders" elected during deployment. However, in the off-duty model the use of such local coordinators is

out of question: they may be unavailable during the event exploration phase. This has implications both on the design of algorithms as well as the achievable performance.

1.3 Our Contribution

This paper initiates the study of a new model for estimating the size of an event, where the affected nodes need to coordinate without the help of the remaining nodes. The task is non-trivial, because nodes do not know the structure of the active component in advance. On the other hand, we aim to make the runtime and the number of messages dependent on the size of the affected node set and not on the size of the network. In this work, we study the tradeoffs between the runtime and the number of messages needed to solve the task.

We begin our investigations with a case study of one-hop networks (i.e., complete graphs, see Section 2). We describe a natural and simple algorithm family Group, where nodes organize in groups of increasing sizes. Subsequently, we describe a randomized Las Vegas approach Rand where nodes seek to "guess" the number of active nodes in order to coordinate. In expectation, the algorithm requires time $O(\log(n/s))$ and $O(n)$ messages. In Section 3, we complement our insights on clique algorithms with lower bounds. We show that our problem requires an understanding of the intriguing interplay of time and message complexity, and use Turán's theorem together with the concept of primary schedules to show that the product of time and message complexity of any deterministic algorithm is at least $\Omega(n \log \log n)$. This proves that our results for clique are optimal up to logarithmic factors.

Section 4 proceeds to examine arbitrary topologies. We first discuss general graph searching techniques combined with waiting techniques, and then introduce a preprocessing scheme, which allows each node to locally and efficiently detect its active neighbors. The complexity of this scheme depends on a graph's arboricity, i.e., the forest cover size. Given this construction, we present the MinID algorithm with time complexity $O(s \log s)$ and message complexity $O(\alpha s \log s)$, i.e., its performance only depends on the event size s and the graph arboricity α.

Motivated by our results for general graphs, we tackle the important case of planar graphs (Section 5), which are known to have constant arboricity. There the time and message complexities of our algorithm MinID are optimal up to an $O(\log s)$ factor. We also show that the message complexity can be improved: using the Planar Separator Theorem, we construct a graph decomposition approach resulting in an algorithm family k-Sep which yields optimal message complexity and non-trivial time.

2 The Clique

To start our scrutinies and to get acquainted with the model, we consider the case of one-hop networks. Note that in such networks, active nodes form only one connected component. Clearly, here a broadcast by all active nodes is already

time-optimal, but it requires $s \cdot (n-1)$ messages. We therefore study a natural class of algorithms called GROUP where nodes organize themselves recursively into groups. Subsequently, we extend our scope to randomized algorithms and study an efficient algorithm RAND which tries to estimate the active set cardinality.

2.1 The Algorithm GROUP

The algorithm GROUP uses an integer parameter $k \in \{2, \dots n\}$. For simplicity of description, we assume that $\log_k n$ is an integer as well. We further assume that node identifiers are written as $(\log_k n)$-digit strings, where each digit is an integer between 0 and $k-1$. This implicitly creates the following hierarchical partitioning of all nodes into clusters. The topmost cluster (on level $\log_k n$) contains all nodes and is divided into k clusters, each consisting of n/k nodes, where cluster i contains all the nodes whose first digit is equal to i. Each of these clusters is also partitioned into k clusters on the basis of the second digit of identifiers. This partitioning proceeds to leafs, which are 0^{th} level clusters, each containing a single node. We call a cluster *active* if it contains at least one active node.

GROUP works in $\log_k n$ phases, where in the i^{th} phase we are dealing with clusters on level i. We inductively require that at the beginning of the phase, there is a *leader* in each i^{th} level active cluster; the leader knows all the active nodes within its cluster and all these nodes know the leader. Thus, at the end of the phase $\log_k n$, all nodes constitute a single cluster and its leader knows the set of all active nodes.

To study what happens in the i^{th} phase, we concentrate on a single $(i+1)^{th}$ level cluster A. (The procedure is performed in all such clusters independently in parallel.) A consists of k i^{th} level clusters, denoted $A_1, A_2, \dots, A_k$, which will merge in this phase. Moreover the leader of A is the node with smallest ID amongst leaders of active clusters A_i. The merging procedure comes in two flavors: parallel (PAR) and sequential (SEQ).

In the PAR variant, a phase lasts two rounds. In the first round, the leaders of clusters A_j broadcast a "hello message" to all nodes from these clusters. All the active nodes among them answer with a message to a leader with the smallest identifier, and this node becomes a leader of the $(i+1)^{th}$ level cluster.

In the SEQ variant, the phase lasts for $k+1$ rounds. For $j \leq k$, in the j^{th} round, the leader of cluster A_j broadcasts a hello message to all nodes from A, provided such a message was not sent already. The nodes which hear the message, answer in the next round, and the leader that transmitted the broadcast becomes a leader of the next higher level cluster, the $(i+1)^{th}$ level cluster.

Theorem 1. *Fix any $1 \leq \ell \leq \log n$. Then there exists a parameterization of the algorithm* GROUP *which solves the problem using $O(\ell)$ rounds and $O(n \cdot \ell \cdot \min\{n^{1/\ell}, s\})$ messages and there exists a parameterization, which solves it using $O(\ell \cdot n^{1/\ell})$ rounds and $O(n \cdot \ell)$ messages.*

Proof. We measure the time and message complexity of the GROUP algorithm using variants PAR and SEQ for all levels and choosing $k = n^{1/\ell}$, i.e. $\log_k n = \ell$.

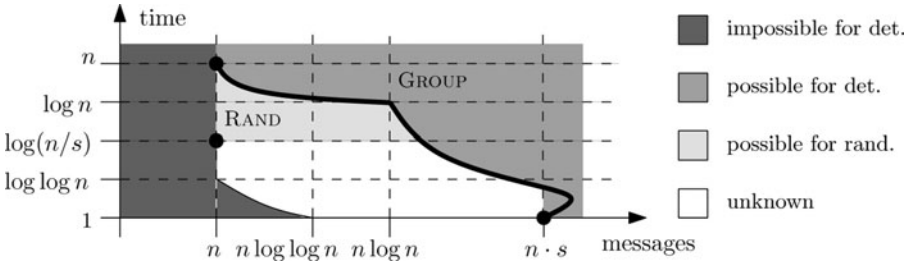


Fig. 1. Performance comparison of different clique algorithms. Black dots represent different algorithms and the black line indicates the trade-offs achievable with the GROUP algorithm. Darker regions represent infeasibility results.

Clearly, in the PAR variant, the algorithm needs $2 \cdot \log_k n = O(\ell)$ rounds. As for the message complexity, we look at a single phase i. In the first round of this phase, each node gets a hello message from at most $\min\{k, s\}$ leaders and sends a reply. Therefore, the algorithm uses at most $2 \cdot \min\{k, s\} \cdot n \cdot \log_k n = O(n \cdot \ell \cdot \min\{n^{1/\ell}, s\})$ messages. The variant SEQ requires $(k+1) \cdot \log_k n = O(\ell \cdot n^{1/\ell})$ rounds. Then, in a single phase, each node gets at most one hello message and answers at most once. Thus, the total number of messages transmitted is at most $2 \cdot n \cdot \log_k n = O(n \cdot \ell)$. □

We observe that the best time×message-product is achieved for $\ell = \log n$, in which case GROUP solves the problem in time $O(\log n)$ using $O(n \log n)$ messages. Note that GROUP can be regarded as a generalization of two graph search techniques: the extreme cases require 1 round or n messages and correspond to parallel or sequential flooding of the graph by active nodes. These trade-offs are depicted in Figure 1.

2.2 Randomized Cardinality Guessing

In this section, we extend our analysis to randomized approaches. The idea behind our algorithm RAND is to approximately "guess" the number of active nodes. For succinctness of the description, we assume that n is a power of 2. RAND proceeds in $\log n + 1$ phases, numbered from 0 to $\log n$, each consisting of two rounds. In the first round of the i^{th} phase, each node — with probability $p_i = 2^i/n$ — broadcasts a hello message to all other nodes. In the second round active nodes reply. After a phase with a broadcast, the algorithm terminates. The Las Vegas algorithm RAND always solves the problem, as in phase $\log n$ each node performs a broadcast (with probability 1).

Theorem 2. *On expectation,* RAND *terminates in* $O(\log(n/s))$ *rounds and uses* $O(n)$ *messages.*

Proof. Let $k = \lceil \log(n/s) \rceil$, i.e., $2^{k-1} < n/s \leq 2^k$. Then, phase k is the first phase in which the broadcast probability of each node reaches $1/s$, i.e., $p_k \in$

$[1/s, 2/s)$. It is sufficient to show that the algorithm makes its first broadcast around phase k, and, in expectation, it makes a constant number of broadcasts.

Let $\mathcal{E}_i$ denote an event that RAND does not finish till phase i (inclusive), i.e., there was no broadcast in phases $1, 2, \ldots, i$. Let τ be a random variable denoting the number of phases of RAND. Then, $\mathbf{E}[\tau] = \sum_{i=1}^{\log n} \Pr[\tau \geq i] = \sum_{i=0}^{\log n - 1} \Pr[\mathcal{E}_i] \leq \sum_{i=0}^{k-1} 1 + \sum_{j=0}^{\log n - k - 1} \Pr[\mathcal{E}_{k+j}]$.

To bound the last term, we first observe that the necessary condition for $\mathcal{E}_i$ is that no node transmits in phase i. Hence, $\Pr[\mathcal{E}_i] \leq (1 - p_i)^s$, and thus for $0 \leq j \leq \log n - k - 1$,

$$\Pr[\mathcal{E}_{k+j}] = \left(1 - \frac{2^{k+j}}{n}\right)^s \leq \left(\frac{1}{\mathrm{e}}\right)^{\frac{2^{k+j}}{n} \cdot s} \leq \mathrm{e}^{-2^j} \ .$$

Therefore, $\mathbf{E}[\tau] \leq k + O(1)$.

Now, we upper-bound the number of transmitted messages. Let X_i be a random variable denoting the number of nodes transmitting in phase i. Then, $\mathbf{E}[X_i | \mathcal{E}_{i-1}] = s \cdot p_i$. The expected total number of transmitted messages is then

$$\begin{aligned}
\mathbf{E}\left[\sum_{i=0}^{\log n} X_i\right] &= \sum_{i=0}^{k+1} \mathbf{E}[X_i | \mathcal{E}_{i-1}] \cdot \Pr[\mathcal{E}_{i-1}] + \sum_{j=1}^{\log n - k - 1} \mathbf{E}[X_{k+j+1} | \mathcal{E}_{k+j}] \cdot \Pr[\mathcal{E}_{k+j}] \\
&\leq \sum_{i=1}^{k+1} s \cdot p_i \cdot 1 \ + \sum_{j=1}^{\log n - k - 1} s \cdot p_{k+j+1} \cdot \mathrm{e}^{-2^j} \\
&\leq 4 \cdot s \cdot p_k + s \cdot p_k \sum_{j=1}^{\infty} \left(2^{j+1} \cdot \mathrm{e}^{-2^j}\right) \\
&= O(s \cdot p_k) = O(1) \ .
\end{aligned}$$

As the expected number of broadcasts is constant, the expected number of messages is $O(n)$. □

3 Lower Bounds

Our bounds from the previous section raise the question of what can and cannot be achieved in distributed event size estimation. Hence, we turn our attention to lower bounds. Clearly, in any graph, an algorithm solving the problem needs $\Omega(\delta)$ rounds, as there exists a pair of active nodes in distance δ and any node has to communicate with both of them. Also, each active node has to send or hear at least one message, and therefore the total communication is at least $\Omega(s)$.

Below we again concentrate on cliques. Fix any deterministic algorithm ALG, and assume that only node i is active. Then i transmits messages in some particular order, which we call a *primary schedule for* i. Note that for any starting set of active nodes, ALG uses the primary schedule for i as long as i does not receive a message from other nodes. For succinctness, we say that ALG *p-sends*

a message in round ℓ, meaning that the primary schedule of ALG sends a message in round ℓ. We say that two nodes *p-contact* if one of them p-sends a message to the other. Using an averaging argument, we may find a pair of nodes which p-contact after transmitting many messages.

Lemma 1. *For any deterministic algorithm for the clique and for any subset of k nodes A, there exists a pair of nodes $v, v' \in A$ which p-contact only after either of them p-sends at least $k/2 - 1$ messages.*

Proof. First, we observe that the total number of messages in all primary schedules is at least $\binom{k}{2}$. Otherwise, there would exist a pair of nodes which never p-contact. In effect, if the algorithm is run on an instance where only these two nodes are active, it cannot solve the problem, as none of these nodes can distinguish between instances where the second node is active or inactive.

For simplicity of the description, we assume that messages are p-sent sequentially. The j-th message of node i receives label j. An edge between node i and i' receives the label which is the minimum of the labels of messages sent from i to i' and from i' to i. It is therefore sufficient to show that there exists an edge with label at least $k/2$. Assume the contrary, i.e., all edges have labels of at most $k/2 - 1$. Then the label of any message would be at most $k/2 - 1$, which would imply that the total number of p-sent messages is $k \cdot (\frac{k}{2} - 1) < \binom{k}{2}$. □

By Lemma 1, it is possible, for any given algorithm, to choose two active nodes in a clique, so that they contact after sending $\Omega(n)$ messages. In a similar way, we may show that the $\Omega(n)$-message bound holds for an arbitrary number of active nodes.

Corollary 1. *The number of messages sent by any deterministic algorithm in a clique is at least $\Omega(n)$.*

Theorem 3. *For any fixed s and n, there exists an n-node graph, such that for any algorithm for this graph, there exists an instance of the problem with s active nodes, on which the algorithm performs $\Omega(n)$ message transmissions.*

Proof. Fix any $s \leq n$. If $s \geq n/2$, then the theorem follows immediately in any graph by the trivial $\Omega(s)$ lower bound.

Otherwise, we assume that $s < n/2$ and we construct a graph, in which a choice of $s-2$ active nodes is already fixed in a way which cannot help meeting the remaining two active nodes. The graph is depicted in Figure 2. Its nodes are partitioned into three sets: a chain S_0 of $s-2$ nodes, and sets S_1, S_2 containing $\lfloor (n-s+2)/2 \rfloor$ and $\lceil (n-s+2)/2 \rceil$ nodes, respectively. Sets S_1 and S_2 form a complete bipartite graph.

The algorithm is run on an instance where all $s-2$ nodes of S_0 are active, and exactly one node $v_i \in S_1$ and one node $v_j \in S_2$ is active. We show that there exists a pair of nodes v_i, v_j which contact after sending $\Omega(n)$ messages.

As v_i knows that all nodes from S_0 are active, its primary schedule is not affected if it contacts or is contacted by a node from S_0. This time we consider only nodes from S_1 and S_2 and messages crossing the edges between these two

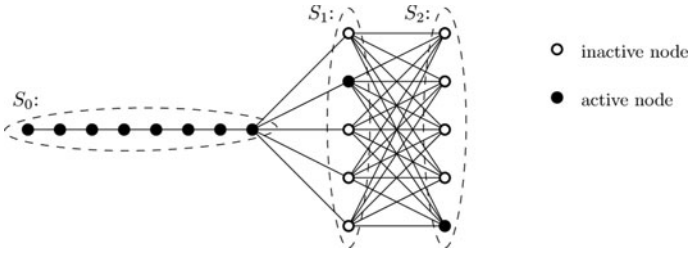


Fig. 2. Illustration of the lower bound for arbitrary number of active nodes

sets. The total number of such messages in all primary schedules have to be at least $|S_1| \cdot |S_2|$. Using the same labeling technique as in the proof of Lemma 1, it is straightforward that there exists an edge between S_1 and S_2 with label at least $f = \frac{|S_1| \cdot |S_2|}{|S_1| + |S_2|}$, as otherwise the total number of messages would be $(|S_1| + |S_2|) \cdot (f - 1) < |S_1| \cdot |S_2|$. Since $f \in \Omega(n)$, the theorem follows. □

The following theorem sheds light on the tradeoff between time and message complexity.

Theorem 4. *Fix any deterministic algorithm* ALG *that solves the problem in a clique using* TIME *rounds and* MSG *messages. Then* TIME·MSG $= \Omega(n \cdot \log\log n)$.

Proof. We assume that $\log\log n \geq 4$. We consider the first t rounds of the nodes' primary schedules, where $t = \log(\log n / \log\log\log n) = \Omega(\log\log n)$.

First, assume that there exists a subset A of $n/2$ nodes, each p-sending less than $n/4$ messages in the first t steps. By Lemma 1, there exists a pair of nodes $v, v' \in A$, which first contact after one of them sends at least $|A|/2 - 1 = n/4 - 1$ messages. Thus, if we start ALG on a graph where only v and v' are active, it takes at least $n/4$ messages and time t.

Hence, in the remaining part of the proof, we assume that there is a set B_0 of at least $n/2$ nodes, each p-sending at least $n/4$ messages within the first t steps.

We create a sequence of sets $\{B_i\}_{i=0}^{t}$, such that B_i is a maximum subset of B_{i-1} with the property that no two nodes of B_i p-send a message to each other in round i. By induction, no node from B_i p-sends a message to another node from B_i within the first i steps. Let $h = \frac{1}{2} \cdot \log\log n$. We show the following property:

> *Assume that for all $i \leq t-1$, the nodes of B_i p-send in total at most $hn/4$ messages in round i. Then for all $i \leq t$, it holds that $|B_i| \geq \frac{n}{2 \cdot (2h)^{2^i - 1}}$.*

We prove the property inductively. The initial case of $i = 0$ holds trivially. Fix any round $i \leq t$. In round $i - 1$ the nodes of B_{i-1} p-sent at most $hn/4$ messages to themselves. Consider a graph on nodes from B_{i-1}, in which an edge exists between a pair of nodes if they contact in round $i - 1$. The average degree in this

graph is $(h{\cdot}n)/(2{\cdot}|B_{i-1}|)$ and by Turán's theorem [1], there exists an independent set B_i of size

$$|B_i| \geq \frac{|B_{i-1}|}{1+\frac{h\cdot n}{2\cdot|B_{i-1}|}} \geq \frac{|B_{i-1}|^2}{h\cdot n} \geq \frac{n^2}{4\cdot(2h)^{2^i-2}\cdot h\cdot n} = \frac{n}{2\cdot(2h)^{2^i-1}} \ .$$

In our context, independence means that the nodes of B_i do not p-contact each other in round i.

Finally, we show how the theorem follows by the property above. If there exists a round $i \leq t-1$ in which nodes of B_i p-send at least $hn/4$ messages, then we run ALG on a graph where only nodes of B_i are active and the theorem follows immediately. Otherwise, B_t contains at least $n/(2\cdot(2h)^{2^t-1}) \geq 2$ nodes. Then, if we run ALG on a graph where only nodes of B_t are active, they do not contact within the first t steps and each of them sends at least $n/4$ messages. □

4 Arbitrary Graphs

In this section, we construct algorithms that perform well on arbitrary graphs. First, we note that it is possible to implement distributed *depth/breadth first search* (DFS/BFS) procedures in our environment.

Lemma 2. *A distributed DFS procedure initiated at a single node finishes in time $O(s)$ using $O(n)$ messages. BFS uses $O(\delta)$ rounds and $O(m)$ messages.*

Proof. A BFS procedure is just a simple flooding and its time and message complexities are straightforward.

As for DFS, we fix a starting node. We say that this node holds the "token": the token indicates the node that would be processed in the centralized DFS. This token is a table of current knowledge about all the nodes: nodes are either known to be active, known to be inactive or have unknown state. During our procedure, the token node tries to forward the token to the neighbor which would be next on the DFS tree. This is done as follows. First, the token node "pings" all its neighbors with unknown state and active neighbors respond immediately. Then like in the centralized DFS algorithm the token is passed to any unvisited active node, and if there is none, the token is sent back to the node from which it came. As DFS proceeds along the DFS tree spanning all active nodes in a single component, it takes time $O(s)$. In the process of gaining knowledge each node changes its state just once, so the number of messages is $O(n)$. □

These procedures are useful if there is a predefined leader. Otherwise, we have to start this procedure at all (active) nodes utilizing some level of parallelism.

Lemma 3. *For arbitrary graphs, for any $1 \leq k \leq n$, there exists an algorithm k-*WAITDFS*, which solves the problem in $O(n^2/k)$ rounds using $O(\min\{k,s\}\cdot n)$ messages. There also exist algorithms* PARDFS*, solving the problem in $O(s)$ rounds using $O(s{\cdot}n)$ messages and* PARBFS *which takes time $O(\delta)$ and performs $O(m\cdot s)$ message transmissions.*

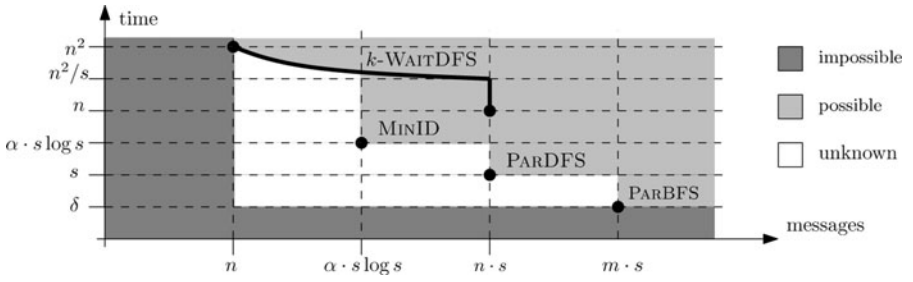


Fig. 3. Performance comparison of different algorithms for arbitrary graphs

Proof. In general, our problem can be solved by running s instances of DFS in parallel. We call this algorithm ParDFS. Obviously, it runs in time $O(s)$ and uses $O(s \cdot n)$ messages.

One way to reduce the number of messages is to have a graded start of the DFS procedures. Of course, as we do not know which nodes are active, we may need to wait for potentially inactive nodes. Concretely, in our algorithm k-WaitDFS, we exploit the fact that the nodes are ordered in the preprocessing stage (i.e. they have IDs from 1 to n). We divide time into $\lceil n/k \rceil$ phases of length $\Theta(n)$. This length is chosen in such a way that for any choice of the active nodes, the worst-case execution of a DFS initiated at any node ends within one phase. In phase i, we define a subset of *busy* nodes. These are nodes with identifiers between $k \cdot (i-1)+1$ and $k \cdot i$, which have not participated in any DFS so far. All nodes which are active and busy start their DFS procedures, transmitting in total at most $O(\min\{k, s\} \cdot n)$ messages in one phase. In the worst-case, the algorithm finishes after $\lceil n/k \rceil$ phases, i.e., after $O(n^2/k)$ rounds.

If we only care about the time complexity, the optimal algorithm initiated by a single node is a BFS (i.e., flooding). Again, we have to cope with an issue of choosing the node which initiates such a search; in the algorithm ParBFS, all nodes perform a BFS concurrently. □

In the remaining part of this section, we first present a technique for efficient neighborhood discovery and later use it in an algorithm MinID, whose performance is output-sensitive and depends, besides s, only on the arboricity of the graph.

4.1 Neighborhood Discovery

So far, the preprocessing stage was used for assigning identifiers to nodes only. In the following, we assume that the nodes pre-compute a list of neighbors they will contact if they get activated by the event.

We employ the concept of the *arboricity* of an arbitrary graph G which is defined as the minimum number of forests α that are required to cover all edges in G. During preprocessing of the network, we compute respective rooted spanning forests $\mathcal{F} = \{F_1, F_2, .., F_\alpha\}$. We note that this decomposition

can be performed in polynomial time [4,10]). For any node v, we define a set $N_v = \{w : \exists\, F_j \in \mathcal{F},\ \text{s.t. } w \text{ is a parent of } v \text{ in } F_j\}$.

In the first round of the event stage, every active node v "pings" all nodes from N_v. At the same time it receives similar probing messages from some of its active neighbors. Pinged active nodes reply in the second round. We observe each active node receives a ping or a reply from each of its active neighbors, and thus learns its active neighborhood.

The neighborhood discovery is performed in two rounds. As each active node v sends $|N_v| \leq \alpha$ test messages followed by the transmission of $|N_v|$ receipts, the total communication complexity is at most $O(\alpha s)$.

4.2 The MinID Algorithm

In the preprocessing phase of MinID, the algorithm assigns identifiers to nodes, discovers the topology, and computes trees $\{F_j\}_{j=1}^{\alpha}$ as described above. In the first two rounds of the event stage, using $O(\alpha s)$ messages, each node learns about its active neighbors. Then a leader election is performed in the way described below. First, we present the algorithm under the assumption that s is known; later we show that this assumption is not critical for our analysis.

The discovery of active components is performed by leader election, where the node with the smallest index distributes its index to everyone else in the component. The algorithm proceeds in $2 \log s$ phases. Initially, each active node v defines its own cluster $C_v = \{v\}$, with v acting as the leader. In due course the number of clusters is reduced, s.t., after at most $2 \log s$ phases a single cluster containing all active nodes in the component is formed. At any time two clusters C_i and C_j are neighbors if there exists an edge (v, w) connecting two active nodes $v \in C_i$ and $w \in C_j$.

We also assume that before entering a new phase each cluster is supported by a spanning tree rooted in the leader. Note that the Euler tour defined on edges of the spanning tree allows to visit all nodes in the cluster in time at most $2s$, e.g., by a token released by the leader. Each cluster C_i is visited by the token three times. During the first visit at each node $v \in C_i$, the token distributes the index i to the entire C_i and all active neighbors of C_i in different clusters. During the second visit, the token collects information about indices of neighboring clusters and it picks the C_j with the smallest index j. If $j < i$, during the third consecutive visit, the token distributes j to all nodes in C_i to inform them that they are now destined for C_j.

Let G_C be a digraph in which the set of nodes is formed of clusters C_i and where there is an arc from C_j to C_i iff nodes of C_i are destined for C_j. A node C_w with in-degree 0 in G_C corresponds to a cluster that during this phase spreads its index to all other clusters reachable from C_w according to the directed connections in G_C. Note also that since the maximum in-degree of nodes in G_C is 1, each cluster with in-degree 1 will receive a new index from exactly one cluster. The process of reindexing is performed by a DFS procedure initiated by the leader in each cluster C_w with in-degree 0 in G_C and it is extended to the nodes of all (not only neighbors) clusters reachable from C_w (according to the

connections in G_C). The three visits along Euler tours followed by reindexing take time $O(s)$. The total communication complexity is $O(\alpha s)$, since every edge is traversed a constant number of times.

It remains to show that during two consecutive phases the number of clusters is reduced by half in non-trivial components (containing at least two clusters). Two cases occur. The first case refers to "amalgamation" of clusters where any cluster either delegates its nodes to some other cluster or is a cluster that provides its index to some other clusters. In this case the number of clusters is reduced by half after the execution of a single phase. The second case refers to clusters whose indices form local minima in G_C. If during the first phase such a cluster C_i has a neighbor C_z that chooses to delegate its nodes to some other cluster C_j we know that $j < i$ and j is adopted as the new index of C_z. And since $j < i$ during the next phase C_i will fall into the first case as a cluster that delegates its nodes to some other cluster. Thus, after at most $2 \log s$ phases exactly one cluster resides in each component. Hence, for a single phase, the total time is $O(s \log s)$ and the total communication $O(\alpha s \log s)$.

Finally, recall that the procedure presented above works under the assumption that the value of s is known in advance. Since this is not the case we take an exponentially increasing sequence of upper bounds $2, 4, .., 2^i, .., 2^{\lceil \log n \rceil}$ on s, and run our algorithm assuming for these consecutive powers of two, until the correct bound on s is found. Note that when the algorithm runs with a wrong assumption on the size of the component, the nodes eventually learn that the component is larger than expected. The nodes in clusters that are about to expand too much are informed by their leaders, and the nodes destined for other clusters, if not contacted by the new leader on time, also conclude that the bound on s is inappropriate. Thus, the process is continued until the appropriate bound on s is found and then it is stopped. Therefore in total the time complexity in a component of size s is bounded by $\sum_{i=1}^{\lceil \log s \rceil} O(2^i \cdot \log 2^i) = O(s \log s)$. Similarly, the total communication is $O(\alpha s \log s)$.

Theorem 5. *In a graph G with arboricity α, the deterministic algorithm* MinID *finishes in $O(s \log s)$ rounds using $O(\alpha s \log s)$ messages.*

5 Planar Graphs

Some of our algorithms work much better for planar graphs. The arboricity of a planar graph is 3 [10]. Thus, MinID runs in time $O(s \log s)$ using $O(s \log s)$ messages.

If we run the DFS and BFS procedures (and their variants) after we perform a neighborhood discovery presented in Section 4.1, then we may decrease the number of used messages to $O(s)$. These procedures are performed in a manner that ignores the existence of non-active nodes. In particular, the number of messages used by ParBFS is then $O(s^2)$.

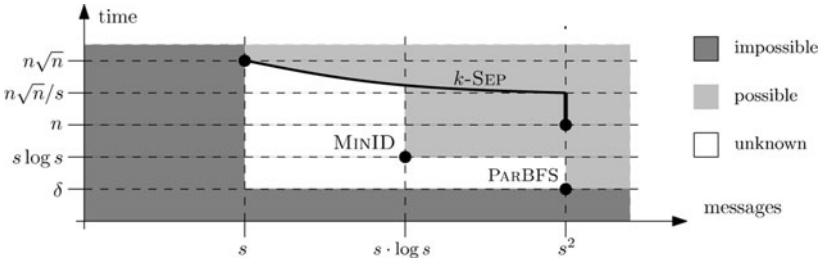


Fig. 4. Performance comparison of different algorithms for planar graphs

Below we present an algorithm k-Sep, which is specially suited for planar graphs. While its runtime is larger than that of MinID, its communication cost is reduced. The performance of the aforementioned algorithms is depicted in Figure 4.

5.1 Hierarchical Decomposition

We start our description from the preprocessing stage. Recall that the planar separator theorem by Lipton and Tarjan [8] enables us to partition any set of nodes V_0 of a planar graph into three disjoint sets: a separator U_0, s.t. $|U_0| \leq c\sqrt{n}$, and sets A_0, B_0 of sizes at most $\frac{2}{3}n$, such that A_0 and B_0 are themselves connected, but there is no edge between them. In the preprocessing stage, this theorem is applied recursively starting from V to produce a binary decomposition tree; if an internal node corresponds to a set $V_0 = U_0 \uplus A_0 \uplus B_0$, then its children correspond to sets A_0 and B_0; each leaf contains a single node.

At the beginning of the event stage, the algorithm Sep performs a neighborhood discovery. Then it proceeds recursively as described below. Sep starts from the root of the tree, which corresponds to three sets $V = V_0 = U_0 \uplus A_0 \uplus B_0$. It initiates $|U_0|$ DFS procedures sequentially, i.e., one after another, starting at vertices of U_0. These procedures are allowed to visit only nodes in V_0; for the execution of each of them we reserve $O(|V_0|)$ rounds. Moreover, a node which already took part in any DFS, does not start its own DFS. Thus, if an active component has a non-empty intersection with U_0, say at a node v, then a DFS in which v participates solves the task in this component. There are possibly other active components. This is where the separating property comes into play: such an active component is contained entirely inside A or inside B, and Sep is run recursively in parallel for these sets.

To bound the number of messages, we observe that each active node participates in exactly one DFS, and thus $O(s)$ messages suffice. To bound the number of rounds, we observe that any separator set contains at most $c\sqrt{n}$ vertices. Further, the execution of a single DFS started within a set V_0 takes time $O(|V_0|)$. Therefore, the number of used rounds is at most $O(c\sqrt{n}) \cdot (n + \frac{2}{3}n + \left(\frac{2}{3}\right)^2 n + \ldots) = O(n \cdot \sqrt{n})$.

Using the same technique of setting the level of parallelism of DFS procedures as in k-WaitDFS, we derive the following theorem.

Theorem 6. *For any $1 \leq k \leq n$, there exists an algorithm k-Sep, which solves the problem in $O(n \cdot \sqrt{n}/k)$ rounds, using $O(\min\{s \cdot k\} \cdot s)$ messages.*

Proof. The algorithm k-Sep works essentially in the same way as the original Sep algorithm, but within a single set $V_0 = U_0 \uplus A_0 \uplus B_0$ corresponding to a tree node, it utilizes some level of parallelism in running DFS procedures. Namely, an original Sep algorithm runs the 1-WaitDFS procedure there, whereas k-Sep runs k-WaitDFS, where each of $|U_0|$ DFS procedures is run for $O(|V_0|)$ steps. As in the analysis of the k-WaitDFS algorithm, this gives us at most $O(\min\{k, s\} \cdot s)$ messages and $O((c \cdot \sqrt{n}/k) \cdot |V_0|)$ rounds.

In total, the algorithm also uses at most $O(\min\{k, s\} \cdot s)$ messages, because it stops after a successful DFS. The total number of rounds is then $O(\sqrt{n}/k) \cdot (n + \frac{2}{3}n + \left(\frac{2}{3}\right)^2 n + \ldots) = O(n \cdot \sqrt{n}/k)$. □

References

1. Alon, N., Spencer, J.: The Probabilistic Method. John Wiley, Chichester (1991)
2. Birk, Y., Keidar, I., Liss, L., Schuster, A., Wolff, R.: Veracity Radius: Capturing the Locality of Distributed Computations. In: Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC), pp. 102–111 (2006)
3. Davison, A.: Laptops as Earthquake Sensors. In: MIT Technology Review (2008)
4. Gabow, H.N., Westermann, H.H.: Forests, Frames, and Games: Algorithms for Matroid Sums and Applications. Algorithmica 7(1), 465–497 (1992)
5. Jacob, R., Richa, A., Scheideler, C., Schmid, S., Taeubig, H.: A Polylogarithmic Time Algorithm for Distributed Self-Stabilizing Skip Graphs. In: Proc. 28th ACM Symposium on Principles of Distributed Computing, PODC (2009)
6. Jiang, C., Dong, G., Wang, B.: Detection and Tracking of Region-Based Evolving Targets in Sensor Networks. In: Proc. 14th International Conference on Computer Communications and Networks, ICCCN (2005)
7. Kim, H.-A., Karp, B.: Autograph: Toward Automated, Distributed Worm Signature Detection. In: Proc. 13th Usenix Security Symposium, Security (2004)
8. Lipton, R.J., Tarjan, R.E.: A Separator Theorem for Planar Graphs. SIAM Journal of Applied Mathematics 36, 177–189 (1979)
9. Mans, B., Schmid, S., Wattenhofer, R.: Distributed Disaster Disclosure. In: Proc. 11th Scandinavian Workshop on Algorithm Theory, SWAT (2008)
10. Nash-Williams, C.S.J.: Edge-disjoint spanning trees of finite graphs. J. of London Mathematical Society 36, 445–450 (1961)
11. Peleg, D.: Distributed Computing: A Locality-sensitive Approach, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics (2000)

Asynchronous Deterministic Rendezvous in Bounded Terrains

Jurek Czyzowicz[1,⋆], David Ilcinkas[2,⋆⋆], Arnaud Labourel[2,⋆⋆,⋆⋆⋆], and Andrzej Pelc[1,†]

[1] Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada
{jurek,pelc}@uqo.ca
[2] LaBRI, CNRS & Université de Bordeaux, 33405 Talence, France
david.ilcinkas@labri.fr, labourel.arnaud@gmail.com

Abstract. Two mobile agents (robots) have to meet in an a priori unknown bounded terrain modeled as a polygon, possibly with polygonal obstacles. Robots are modeled as points, and each of them is equipped with a compass. Compasses of robots may be incoherent. Robots construct their routes, but the actual walk of each robot is decided by the adversary that may, e.g., speed up or slow down the robot. We consider several scenarios, depending on three factors: (1) obstacles in the terrain are present, or not, (2) compasses of both robots agree, or not, (3) robots have or do not have a map of the terrain with their positions marked. The cost of a rendezvous algorithm is the worst-case sum of lengths of the robots' trajectories until their meeting. For each scenario we design a deterministic rendezvous algorithm and analyze its cost. We also prove lower bounds on the cost of any deterministic rendezvous algorithm in each case. For all scenarios these bounds are tight.

Keywords: mobile agent, rendezvous, deterministic, polygon, obstacle.

1 Introduction

The problem and the model. Two mobile agents (robots) modeled as points starting at different locations of an a priori unknown bounded terrain have to meet. The terrain is represented as a polygon possibly with a finite number of polygonal obstacles. We assume that the boundary of the terrain is included in it. Thus, formally, a terrain is a set $\mathcal{P}_0 \setminus (\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_k)$, where $\mathcal{P}_0$ is a closed polygon and $\mathcal{P}_1, \ldots, \mathcal{P}_k$ are disjoint open polygons included in $\mathcal{P}_0$. We assume

⋆ Partially supported by NSERC discovery grant.
⋆⋆ Partially supported by the ANR project ALADDIN, the INRIA project CEPAGE and by a France-Israel cooperation grant (Multi-Computing project).
⋆⋆⋆ This work was done during this author's stay at the Université du Québec en Outaouais as a postdoctoral fellow.
† Partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 72–85, 2010.

that a robot knows if it is at an interior or at a boundary point, and in the latter case it is capable of walking along the boundary in both directions (i.e., it knows the slope(s) of the boundary at this point). However, a robot cannot sense the terrain or the other robot at any vicinity of its current location. Meeting (rendezvous) is defined as the equality of points representing robots at some moment of time.

We assume that each robot has a unit of length (not necessarily the same for the two agents) and a compass. Compasses of robots may be incoherent, however we assume that robots have the same (clockwise) orientation of their system of coordinates. An additional tool, which may or may not be available to the robots, is a map of the terrain. The map available to a robot is scaled (i.e., it accurately shows the distances), distinguishes the starting positions of this robot and the other one, and is oriented according to the compass of the robot. (Hence maps of different robots may have different North.)

All our considerations concern deterministic algorithms. The crucial notion is the *route* of the robot which is a finite polygonal path in the terrain. The adversary initially places a robot at some point in the terrain. The robot constructs its route in steps in the following way. In every step, the robot starts at some point v; in the first step, v is the starting point chosen by the adversary. The robot chooses a direction α, according to its compass, and a distance x. If the segment of length x in direction α starting in v does not intersect the boundary of the terrain, the step ends when the robot reaches point u at distance x from v in direction α. Otherwise, the step ends at the closest point of the boundary in direction α. If the starting point v in a step is in a segment of the boundary of the terrain, the robot has also an option (in this step) to follow this segment of the boundary in any of the two directions until its end or for some given distance along it. Steps are repeated until rendezvous, or until the route of the robot is completed.

We consider the *asynchronous* version of the rendezvous problem. The asynchrony of the robots' movements is captured by the assumption that the actual walk of each robot is decided by the adversary: the movement of the robot can be at arbitrary speed, the robot may sometimes stop or go back and forth, as long as the walk of the robot in each segment of its route is continuous, does not leave it and covers all of it.[1] More formally, the route in a terrain is a sequence $(S_1, S_2, \ldots, S_k)$ of segments, where $S_i = [a_i, a_{i+1}]$ is the segment corresponding to step i. In our algorithms the route is always finite. This means that the robot stops at some point, regardless of the moves of the other robot. We now describe the walk f of a robot on its route. Let $R = (S_1, S_2, \ldots, S_k)$ be the route of a robot. Let $(t_1, t_2, \ldots, t_{k+1})$, where $t_1 = 0$, be an increasing sequence of reals, chosen by the adversary, that represent points in time. Let $f_i : [t_i, t_{i+1}] \to [a_i, a_{i+1}]$ be any continuous function, chosen by the adversary, such that $f_i(t_i) = a_i$ and

[1] Notice that this definition of the adversary is very strong. In fact, all our positive results (algorithms and their complexity) are valid even with this powerful adversary, and our negative results hold even for a weaker adversary that can only speed up or slow down the robot, without moving it back.

$f_i(t_{i+1}) = a_{i+1}$. For any $t \in [t_i, t_{i+1}]$, we define $f(t) = f_i(t)$. The interpretation of the walk f is as follows: at time t the robot is at the point $f(t)$ of its route and after time t_{k+1} the robot remains inert. This general definition of the walk and the fact that it is constructed by the adversary capture the asynchronous characteristics of the process. Throughout the paper, *rendezvous* means deterministic asynchronous rendezvous.

Robots with routes R and R' and with walks f and f' meet at time t, if points $f(t)$ and $f'(t)$ are equal. A rendezvous is guaranteed for routes R and R', if the robots using these routes meet at some time t, regardless of the walks chosen by the adversary. The trajectory of a robot is the sequence of segments on its route until rendezvous. (The last segment of the trajectory of a robot may be either the last segment of its route or any of its segments or a portion of it, if the other robot is met there.) The cost of a rendezvous algorithm is the worst case sum of lengths of segments of trajectories of both robots, where the worst case is taken over all terrains with the considered values of parameters, and all adversarial decisions.

We consider several scenarios, depending on three factors: (1) obstacles in the terrain are present, or not, (2) compasses of both robots agree, or not, (3) robots have or do not have a map of the terrain. Combinations of the presence or absence of these factors give rise to eight scenarios. For each scenario we design a deterministic rendezvous algorithm and analyze its cost. We also prove lower bounds on the cost of any deterministic rendezvous algorithm in each case. For all scenarios these bounds are tight.

One final clarification has to be made. For all scenarios except those with incoherent compasses and the presence of obstacles (regardless of the availability of a map), robots may be anonymous, i.e., they execute identical algorithms. By contrast, with the presence of obstacles and incoherent compasses, anonymity would preclude feasibility of rendezvous in some situations. Consider a square with one square obstacle positioned at its center. Consider two robots starting at opposite (diagonal) corners of the larger square, with compasses pointing to opposite North directions. If they execute identical algorithms and walk at the same speed, then at each time they are in symmetric positions in the terrain and hence rendezvous is impossible. The only way to break symmetry for a deterministic rendezvous in this case is to equip the robots with distinct labels (which are positive integers). Hence, this is the assumption we make for the scenarios with the presence of obstacles and incoherent compasses (both with and without a map). For any label μ, we denote by $|\mu|$ the length of the binary representation of the label, i.e., $|\mu| = \lfloor \log \mu \rfloor + 1$.

Our results. The cost of our algorithms depends on some of the following parameters (different parameters for different scenarios, see the discussion in Section 4): D is the distance between starting positions of robots in the terrain (i.e., the length of a shortest path between them included in the terrain), P is the perimeter of the terrain, (i.e., the sum of perimeters of all polygons $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_k$), x is the largest perimeter of an obstacle, and l and L are the smaller and larger labels of the two robots, respectively, for the two scenarios

that require different labels, as remarked above., i.e., for the scenarios with the presence of obstacles and incoherent compasses.

Our rendezvous algorithms rely on two different ideas: either meeting in a uniquely defined point of the terrain, or meeting on a uniquely defined cycle. It turns out that a uniquely defined point can be found in all scenarios except those with the presence of obstacles and incoherent compasses. Apart from this exception even anonymous robots can meet. On the other hand, with the presence of obstacles and incoherent compasses, such a uniquely defined point may not exist, as witnessed by the above quoted example of a square with one square obstacle positioned at its center. For these scenarios we resort to the technique of meeting at a common cycle, breaking symmetry by different labels of robots.

We first summarize our results concerning rendezvous when each of the robots is equipped with a map showing its own position and that of the other robot. If compasses of the robots are coherent, then we show a rendezvous algorithm at cost D, which is clearly optimal. Otherwise, and if the terrain does not contain obstacles, then we show an algorithm whose cost is again D, and hence optimal. Finally, with incoherent compasses in the presence of obstacles, we show a rendezvous algorithm at cost $O(D|l|)$; in the latter scenario we show that cost $\Omega(D|l|)$ is necessary for some terrains.

Our results concerning rendezvous without a map are as follows. If compasses of the robots are coherent, then we show a rendezvous algorithm at cost $O(P)$. We also show a matching lower bound $\Omega(P)$ in this case. If compasses of the robots are incoherent, but the terrain does not contain obstacles, then we show a rendezvous algorithm at cost $O(P)$ and again a matching lower bound $\Omega(P)$. Finally, in the hardest of all scenarios (presence of obstacles, incoherent compasses and no map) we have a rendezvous algorithm at cost $O(P + x|L|)$ and a matching lower bound $\Omega(P + x|L|)$. Table 1 summarizes our results. Due to lack of space, some proofs are removed.

Table 1. Summary of results

Rendezvous with a map			Rendezvous without a map		
obstacles \ compasses	coherent	incoherent	obstacles \ compasses	coherent	incoherent
no	D	D	no	$\Theta(P)$	$\Theta(P)$
yes		$\Theta(D\|l\|)$	yes		$\Theta(P + x\|L\|)$

Related work. The rendezvous problem was first described in [24]. A detailed discussion of the large literature on rendezvous can be found in the excellent book [4]. Most of the results in this domain can be divided into two classes: those considering the geometric scenario (rendezvous in the line, see, e.g., [16,25], or in the plane, see, e.g., [7,8]), and those discussing rendezvous in graphs, e.g., [2,5]. Some of the authors, e.g., [2,3,6] consider the probabilistic scenario where inputs and/or rendezvous strategies are random. Randomized rendezvous strategies use

random walks in graphs, which were thoroughly investigated and applied also to other problems, such as, e.g., graph traversing [1]. A generalization of the rendezvous problem is that of gathering [15,18,19], when more than two robots have to meet in one location.

If graphs are unlabeled, deterministic rendezvous requires breaking symmetry, which can be accomplished either by allowing marking nodes or by labeling the robots. Deterministic rendezvous with anonymous robots working in unlabeled graphs but equipped with tokens used to mark nodes was considered e.g., in [21]. In [26] the authors studied the task of gathering many robots with unique labels. In [14,20,27] deterministic rendezvous in graphs with labeled robots was considered. However, in all the above papers, the synchronous setting was assumed. Asynchronous gathering under geometric scenarios has been studied, e.g., in [11,15,22] in different models than ours: robots could not remember past events, but they were assumed to have at least partial visibility of the scene. The first paper to consider deterministic asynchronous rendezvous in graphs was [12]. The authors concentrated on complexity of rendezvous in simple graphs, such as the ring and the infinite line. They also showed feasibility of deterministic asynchronous rendezvous in arbitrary finite connected graphs with *known* upper bound on the size. Further improvements of the above results for the infinite line were proposed in [25]. Gathering many robots in a graph, under a different asynchronous model and assuming that the whole graph is seen by each robot, has been studied in [18,19].

2 Rendezvous with a Map

We start by describing the following procedure that finds a unique shortest path from the starting position of one robot to the other. The procedure works in all scenarios in which robots have a map of the terrain with their positions indicated.

Procedure path `UniquePath`(point v, point w)
1 point $u := v$; path $p := \{v\}$;
2 $\mathcal{S} = \{p_s \mid p_s$ is a shortest path between v and $w\}$;
3 **while** $(u \neq w)$ **do**
4 $\mathcal{U}$:=all paths p_s of $\mathcal{S}$ such that the first segment of the subpath of p_s leading from u to w is the first in clockwise order around u starting from the direction vw;
5 $p' := \bigcap_{p_s \in \mathcal{U}} p_s$;
6 extend p with the connected part of p' containing u;
7 $u :=$ new end of path p;
8 **return** p;

Lemma 1. *Procedure* `UniquePath` *computes a unique shortest path from v to w, independent of the robot computing it.*

2.1 Coherent Compasses

If robots have a map and coherent compasses, then they can easily agree on one of their two starting positions and meet at this point at cost D, which is optimal. This is done by the following Algorithm RVCM (rendezvous with a map and coherent compasses).

Algorithm *RVCM*
Let v be the northernmost of the two starting positions of the robots. If both robots have the same latitude, let v be the easternmost of them. Let w be the other starting position. The robot starting at v remains inert. The robot starting at w computes the path $p = \texttt{UniquePath}(w, v)$ and moves along p until v.

Theorem 1. *Algorithm RVCM guarantees rendezvous at cost D, for any two robots with a map and coherent compasses, in any terrain.*

2.2 Incoherent Compasses

Terrains without obstacles
In an empty polygon there is a unique shortest path between starting positions of the robots [9], and robots with a map can meet in the middle of this path at cost D, which is optimal. This is done by Algorithm RVM (rendezvous with a map, without obstacles).

Algorithm *RVM*
The robot computes the (unique) shortest path between the starting positions of the two robots. Then, it moves along this shortest path until the middle of it.

Theorem 2. *Algorithm RVM guarantees rendezvous at cost D for any two robots with a map, in any terrain without obstacles.*

Terrains with obstacles
This is the first of the two scenarios where robots cannot always predetermine a meeting point. Therefore they compute a common embedding of a ring on which they are initially situated, and then each robot executes the rendezvous procedure from [12] for this ring. For the sake of completeness, this procedure is briefly described below. It consists of two parts: Label Transformation and Label Execution. The Label Transformation part takes the label μ of an agent and produces the label μ^* in the following way. First produce label μ' consisting of a string of $|\mu|$ zeros, followed by a 1 and then followed by the string μ. The label μ^*, called the *transformed label* of the agent, is obtained by replacing in μ' each 0 by 01 and each 1 by 10. The Label Execution part is divided into phases numbered 1,2,... For a given agent, we define the execution of bit 0 (resp. 1) in

phase a as performing 3^a steps left (resp. right), according to the agent's local orientation. For an agent with label μ, phase a consists of consecutive executions of all bits of μ^* from left to right.

Using the above procedure, rendezvous with a map, with obstacles is performed by the following Algorithm RVMO. Recall that in this scenario robots have distinct labels, hence the procedure from [12] can be applied. Rendezvous is guaranteed to occur on the ring, but the meeting point depends on the walks of the robots determined by the adversary.

Algorithm *RVMO*
Phase 1: computation of the embedding[1] $\mathcal{R}$ **of a ring of size** 4.
Let v be the starting position of the robot and let w be the starting position of the other robot. The robot computes the embedding $\mathcal{R}$ of a ring, composed of four nodes v, a, w and b, where a is the midpoint of `UniquePath`(v, w), b is the midpoint of `UniquePath`(w, v), and the four edges are the respective halves of these paths.
Phase 2: rendezvous on $\mathcal{R}$**.**
This phase consists in applying the above described rendezvous procedure from [12] for ring $\mathcal{R}$, whose size (four) is known to the robots.

[a] This embedding is not necessarily homeomorphic with a circle, it may be degenerate.

Theorem 3. *Algorithm RVMO guarantees rendezvous at cost* $O(D|l|)$ *for arbitrary two robots with a map, in any terrain.*

The following lower bound shows that the cost of Algorithm RVMO cannot be improved for some terrains. Indeed, it implies that for all $D > 0$, there exists a polygon with a single obstacle, for which the cost of any rendezvous algorithm for two robots, starting at distance D, is $\Omega(D|l|)$.

Theorem 4. *For any rendezvous algorithm A, for any $D > 0$, and for any integers $k_2 \geq k_1 > 0$, there exist two labels l_1 and l_2 of lengths at most k_1 and at most k_2, respectively, and a polygon with a single obstacle of perimeter $2D$, such that algorithm A executed by robots with labels l_1 and l_2 starting at distance D, requires cost $\Omega(Dk_1)$. This holds even if the two robots have a map.*

3 Rendezvous without a Map

3.1 Coherent Compasses

It turns out that robots can recognize the outer boundary of the terrain even without a map. Hence, if their compasses are coherent, they can identify a uniquely defined point on this boundary and meet in this point. This is done by Algorithm RVC (rendezvous with coherent compasses) at cost $O(P)$.

Algorithm *RVC*
From its starting position v, the robot follows the half-line α pointing to the North, as far as possible. When it hits the boundary of a polygon $\mathcal{P}$ (i.e., either the external boundary of the terrain or the boundary of an obstacle), it traverses the entire boundary of $\mathcal{P}$. Then, it computes the point u which is the farthest point from v in $\mathcal{P} \cap \alpha$. It goes around $\mathcal{P}$ until reaching u again and progresses on α, if possible. If this is impossible, the robot recognizes that it went around the boundary of $\mathcal{P}_0$. It then computes the northernmost points in $\mathcal{P}_0$. Finally, it traverses the boundary of $\mathcal{P}_0$ until reaching the easternmost of these points.

Theorem 5. *Algorithm RVC guarantees rendezvous at cost $O(P)$ for any two robots with coherent compasses, in any terrain.*

The following lower bound shows that the cost of Algorithm RVC is asymptotically optimal, for some polygons even without obstacles. This lower bound $\Omega(P)$ holds even if the distance D between starting positions of robots is bounded and if their compasses are coherent.

Theorem 6. *There exists a polygon of an arbitrarily large perimeter P, for which the cost of any rendezvous algorithm for two robots with coherent compasses starting at any distance $D > 0$, is $\Omega(P)$.*

Proof. Consider the polygon $\mathcal{P}'$ obtained by attaching to each side of a regular k-gon, whose center is at distance $D/8$ from its boundary, a rectangle of length $3D/8$ and of height equal to the side length of the k-gon. The polygon $\mathcal{P}$ is the polygon obtained by gluing two copies of $\mathcal{P}'$ by the small side of one of the rectangles, as depicted in Fig. 1. Let P be the perimeter of the polygon $\mathcal{P}$. We choose $k = \Theta(P/D)$. There are two types of rectangles in $\mathcal{P}$, two *passing* ones (they share one side) and the $2k - 2$ *normal* ones.

Consider all rotations of the polygon $\mathcal{P}$ around its center of symmetry by angles $2\pi i/k$, for $i = 0, \ldots, k-1$. We will prove that any deterministic rendezvous algorithm requires cost $\Omega(P)$ in at least one of the rotated polygons. Each robot

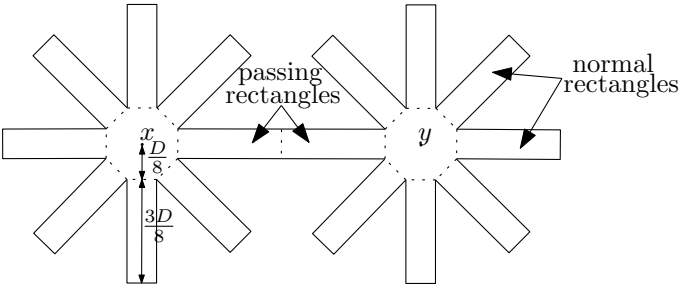


Fig. 1. Polygon $\mathcal{P}$

starts in the center of a different k-gon. We say that a robot has *penetrated* a rectangle if it has moved at distance $D/8$ inside the rectangle. In order to accomplish rendezvous, at least one robot has to penetrate a passing rectangle. Each time one robot penetrates a rectangle, the adversary chooses a rotation, so that all previously penetrated rectangles, including the current one, are normal rectangles. This choice is coherent with the knowledge previously acquired by the robots, since normal rectangles are undistinguishable from each other and a robot needs to penetrate a rectangle in order to distinguish its type. Hence, the two robots have to penetrate a total of $k-1$ rectangles before the adversary cannot rotate the figure to prevent the penetration of a passing rectangle. It follows that at least one of the robots has to traverse a total distance of $\Omega(kD) = \Omega(P)$ before meeting. □

3.2 Incoherent Compasses

Terrains without obstacles

In this section, we use the notion of medial axis, proposed by Blum [10], to define a unique point of rendezvous inside the terrain. Observe that we cannot use the centroid for the rendezvous point since, as we also consider non-convex terrains, the centroid is not necessarily inside the terrain. The *medial axis* $M(\mathcal{P})$ of a polygon $\mathcal{P}$ is defined as the set of points inside $\mathcal{P}$ which have more than one closest point on the boundary of $\mathcal{P}$. Actually, $M(\mathcal{P})$ is a planar tree contained in $\mathcal{P}$, in which nodes are linked by either straight-line segment or arcs of parabolas [23]. We define the *medial point* of a polygon $\mathcal{P}$ as either the central node of $M(\mathcal{P})$ or the middle of the central edge of $M(\mathcal{P})$, depending on whether $M(\mathcal{P})$ has a central node or a central edge. Remark that the medial point of $\mathcal{P}$ is unique and is inside $\mathcal{P}$. The medial axis of a polygon $\mathcal{P}$ can be computed as in [13]. Algorithm RV (rendezvous without obstacles, without a map and with possibly incoherent compasses) determines the unknown (empty) polygon and guarantees meeting in its medial point.

Algorithm *RV*
At its starting position, the robot chooses an arbitrary half-line α which it follows until it hits the boundary of the polygon $\mathcal{P}_0$. It traverses the entire boundary of $\mathcal{P}_0$ and computes the medial point v of $\mathcal{P}_0$. Then, it moves to v by a shortest path and stops.

Theorem 7. *Algorithm RV guarantees rendezvous at cost $O(P)$ for any two robots, in any terrain without obstacles.*

The lower bound from Theorem 6 shows that the cost of Algorithm RV cannot be improved for some polygons.

Terrains with obstacles

Our last rendezvous algorithm, Algorithm *RVO*, works for the hardest of all scenarios: rendezvous with obstacles, no map, and possibly incoherent compasses. Here again it may be impossible to predetermine a meeting point. Thus robots

identify a common cycle and meet on this cycle. The difference between the present setting and that of Algorithm $RVMO$, where a map was available, is that now robots may start outside of the common cycle and have to reach it before attempting rendezvous on it. (Hence, in particular, the robots cannot use directly the procedure for rendezvous in a ring from [12], as was done in Algorithm $RVMO$.) Also the common cycle is different: rather than being composed of two shortest paths between initial positions of the robots (a map seems to be needed to find such paths), it is the boundary of a (possible) obstacle $\mathcal{O}$ in which the medial point of the outer polygon is hidden. These changes have consequences for the cost of the algorithm. The fact that the medial point of the outer polygon has to be found and the obstacle $\mathcal{O}$ has to be reached is responsible for the summand P in the cost. The only bound on the perimeter of this obstacle is x. Finally, the fact that the adversary may delay the robot with the smaller label and force the other robot to make its tours of obstacle $\mathcal{O}$ before the robot with the smaller label even reaches the obstacle, is responsible for the summand $x|L|$, rather than $x|l|$, in the cost.

A *cycle* is a polygonal path whose both extremities are the same point. A *tour* of a cycle $\mathcal{C}$ is any sequence of all the segments of $\mathcal{C}$ in either clockwise or counterclockwise order starting from a vertex of $\mathcal{C}$. By extension, a *partial tour* of $\mathcal{C}$ is a path which is a subsequence of a tour of $\mathcal{C}$ with the first or the last segment of the subsequence possibly replaced by a subsegment of it.

Algorithm RVO

Phase 1: Computation of the medial point of $\mathcal{P}_0$

At its starting position z, the robot chooses an arbitrary half-line α which it follows as far as possible. When it hits the boundary of a polygon $\mathcal{P}$, it traverses the entire boundary of $\mathcal{P}$. Then, it computes the point w which is the farthest point from z in $\mathcal{P} \cap \alpha$. It goes around $\mathcal{P}$ until reaching w again and progresses on α, if possible. If this is impossible, the robot recognizes that it went around the boundary of $\mathcal{P}_0$. The robot computes the medial point v of $\mathcal{P}_0$.

Phase 2: Moving to the medial point of $\mathcal{P}_0$

Let u be the current position of the robot. The robot follows the segment $\overline{uv}$ as far as possible. Similarly as in the first phase of the algorithm, if the robot hits a polygon $\mathcal{P}$, it traverses the entire boundary of $\mathcal{P}$. Then, it computes the point w which is the farthest point from u in $\mathcal{P} \cap \overline{uv}$. It goes around $\mathcal{P}$ until reaching w again and progresses on α, if possible. If this is impossible and if the point v has not been reached, the robot recognizes that v is inside an obstacle $\mathcal{O}$, and executes phase 3. If the robot reaches v, it does not enter phase 3 of the algorithm and stops.

Phase 3: Rendezvous around the medial obstacle of the terrain

The robot goes around the obstacle $\mathcal{O}$ until it reaches a vertex s. The robot produces the modified label μ^* consisting of the binary representation of the label μ of the robot followed by a 1 and then followed by $|\mu|$ zeros. This phase consists of $|\mu^*|$ stages. In stage i, the robot completes two tours of the boundary of $\mathcal{O}$, starting and ending in s, clockwise if the i-th bit of μ^* is 1 and counterclockwise otherwise.

Let $\overline{u_1u_2}$ and $\overline{u_2u_3}$ be consecutive segments in clockwise order (resp. counterclockwise order) of a cycle. For a given walk f of a robot a, we say that the robot *traverses in a clockwise way* (resp. *in a counterclockwise way*) a vertex u_2 of a cycle at time t if $f(t) = u_2$ and there exist positive reals ϵ_1 and ϵ_2 and points y and z such that $y = f(t - \epsilon_1)$ is an internal point of $\overline{u_1u_2}$, $z = f(t + \epsilon_2)$ is an internal point of $\overline{u_2u_3}$ and the robot walks in $\overline{u_1u_2} \cup \overline{u_2u_3}$ during the time period $[t - \epsilon_1, t + \epsilon_2]$.

Before establishing the correctness and cost of Algorithm RVO, we need to show the following two lemmas.

Lemma 2. *Consider two robots on cycle $\mathcal{C}$. Suppose that one robot executes a tour of $\mathcal{C}$ in some sense of rotation, starting and ending in v. If during the same period of time, the other robot either traverses v for the first time in the other sense of rotation or does not traverse it at all, then the two robots meet.*

Lemma 3. *Consider two robots on a cycle $\mathcal{C}$ and let $k \geq 0$ be an integer. If a robot executes either a partial tour of $\mathcal{C}$ followed by at most k tours of $\mathcal{C}$, or at most k tours of $\mathcal{C}$ followed by a partial tour of $\mathcal{C}$, while the second robot executes $k + 2$ tours of $\mathcal{C}$, then the two robots meet.*

Theorem 8. *Algorithm RVO guarantees rendezvous at cost $O(P + x|L|)$ for any two robots in any terrain for which x is the largest perimeter of an obstacle.*

Proof. Let a_1 and a_2 be the two robots that have to meet. The first phase of the algorithm that consists in reaching $\mathcal{P}_0$ and making the tour of the boundary of $\mathcal{P}_0$ costs at most $3P$, since the boundary of each polygon of the terrain is traversed at most twice and the total length of parts of α inside the terrain is at most P. For the same reason as in phase 1, the total cost of phase 2 is at most $3P$.

If the medial point of $\mathcal{P}_0$ is inside the terrain, then the robots meet at the end of phase 2 at total cost of at most $12P$. Otherwise, both robots eventually enter phase 3 of the algorithm and they are on the boundary of the obstacle $\mathcal{O}$ containing the medial point of $\mathcal{P}_0$. The cost follows from the fact that each robot travels a distance $O(x|L|)$ in phase 3. Indeed, each robot executes at most $2|L| + 1$ stages and each stage costs at most $2x$. Hence it remains to show that rendezvous occurs in this case as well.

Assume for contradiction that the two robots never meet. Notice that the modified label l^* cannot be the suffix of the modified label L^*. Indeed, if $|l^*| = |L^*|$ then the two labels are different since $l \neq L$, and otherwise the second part of l^*, consisting of 1 followed by $|l|$ zeros, cannot be the suffix of L^*. Hence, there exists an index i such that the $(|l^*| - i)$-th bit of l^* differs from the $(|L^*| - i)$-th bit of L^*. We call *important* stages the $(|l^*| - i)$-th stage of the robot with label l and the $(|L^*| - i)$-th stage of the robot with label L.

For $j = 1, 2$, let t_j be the moment when robot a_j enters its important stage and let t' be the first moment when both robots have finished the execution of the algorithm. Suppose by symmetry that $t_1 \leq t_2$, i.e., robot a_1 was the first to enter its important stage. Then a_2 must have entered its important stage during the

first tour of the important stage of a_1. Otherwise, robot a_2 would have completed $2i + 2$ tours between t_2 and t', while robot a_1 would have completed at most $2i + 1$ tours. Hence, the two robots would have met in view of Lemma 3. Hence, from the time t_2, robot a_2 completes one tour in some sense of rotation, starting and ending at a vertex v, while robot a_1 either traverses v for the first time in the other sense of rotation or does not traverse it at all. Hence by Lemma 2, the two robots meet. □

The following result gives a lower bound matching the cost of Algorithm RVO.

Theorem 9. *There exist terrains for which the cost of any rendezvous algorithm is* $\Omega(P + x|L|)$. *This holds for arbitrarily small* $D > 0$.

4 Discussion of Parameters

We presented rendezvous algorithms, analyzed their cost and proved matching lower bounds in all considered scenarios. However, it is important to note that the formulas describing the cost depend on the chosen parameters in each case. All our results have the following form. For a given scenario we choose some parameters (among D, P, x, l, L), show an algorithm whose cost in any terrain is $O(f)$, where f is some simple function of the chosen parameters, and then prove that for some class of terrains any rendezvous algorithm requires cost $\Omega(f)$, which shows that the complexity of our algorithm cannot be improved in general, for the chosen parameters.

This yields the question which parameters should be chosen. In the case of complexities D and $\Theta(P)$, this choice does not seem controversial, as here D and P are very natural parameters, and the only ones in these simple cases. However, for the two scenarios with incoherent compasses and with the presence of obstacles, there are several other possible parameters, and their choice may raise a doubt. As mentioned in the introduction, in these two scenarios, distinct labels of robots are necessary to break symmetry, since rendezvous is impossible for anonymous robots. Hence any rendezvous algorithm has to use labels l and L as inputs, and thus the choice of these labels as parameters seems natural. By contrast, the choice of parameter x may seem more controversial. Why do we want to express the cost of a rendezvous algorithm in terms of the largest perimeter of an obstacle? Are there other natural choices of parameter sets? What are their implications?

Let us start by pondering the second question. It is not hard to give examples of other natural choices of parameters for the two scenarios with incoherent compasses and with the presence of obstacles. For example, in the hardest scenario (without a map), we could drop parameter x and try to express the cost of the same Algorithm RVO only in terms of D, P, l, and L. Since $x \leq P$, we would get $O(P|L|)$ instead of $O(P + x|L|)$. Incidentally, as in our lower bound example of terrains we have $x = \Theta(P)$, this new complexity $O(P|L|)$ is optimal for the same reason as the former one.

Another possibility would be adding, instead of dropping a parameter. We could, for example, add the parameter P_e which is the length of the external perimeter of the terrain, i.e., the perimeter of polygon $\mathcal{P}_0$. Then it becomes natural to modify Algorithm RVO as follows. The first two phases are the same. In the third phase, the robot goes around obstacle $\mathcal{O}$ and compares its perimeter to P_e. If the perimeter of $\mathcal{O}$ is smaller (or equal), then the algorithm proceeds as before, and if it is larger, then the robot goes back to the boundary of $\mathcal{P}_0$ and executes Phase 3 on this boundary instead of the boundary of $\mathcal{O}$. The new algorithm has complexity $O(P + \min(x, P_e)|L|)$. Its complexity is again optimal because in our lower bound example we can choose the parameter $y = \min(x, P_e)$ and enlarge the largest of the two boundaries by lengthy but thin zigzags. Thus we can preserve the lower bound $\Omega(P + \min(x, P_e)|L|)$, even when x and P_e differ significantly.

The reason why we chose parameters D, P, l, L, and x instead of just D, P, l and L, is that complexity $O(P+x|L|)$ shows a certain continuity of the complexity of Algorithm RVO with respect to the sizes of obstacles: when the largest obstacle decreases, this complexity approaches $O(P)$ and it becomes $O(P)$ if there are no obstacles. In this case our algorithm coincides with Algorithm RV. This is not the case with complexity $O(P|L|)$. On the other hand, this choice coincides with $O(P + \min(x, P_e)|L|)$ in many important cases, for example for convex obstacles (as then we have $x < P_e$).

It is then natural to ask what happens if we add parameter x in the scenario with incoherent compasses and with the presence of obstacles but with the map. Obviously we could still use Algorithm RVO and get complexity $O(P + x|L|)$. However, our lower bound argument in this scenario gives in fact only $\Omega(D + \min(x, D)|l|)$. In our example we had $D = \Theta(x)$ but we only get $\Omega(D + x|l|)$ even if D is much larger than x. On the other hand, if D is much smaller than x, we can only get the lower bound $\Omega(D|l|)$ because it matches the complexity of $RVMO$ in this case. Hence it is natural to ask if there exists a rendezvous algorithm with cost $O(D + \min(x, D)|l|)$ for arbitrary terrains in this scenario. We leave this as an open question.

References

1. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proc. Annual Symposium on Foundations of Computer Science FOCS 1979, pp. 218–223 (1979)
2. Alpern, S.: The rendezvous search problem. SIAM J. on Control and Optimization 33, 673–683 (1995)
3. Alpern, S.: Rendezvous search on labelled networks. Naval Reaserch Logistics 49, 256–274 (2002)
4. Alpern, S., Gal, S.: The theory of search games and rendezvous. Kluwer Academic Publ., Dordrecht (2002)
5. Alpern, J., Baston, V., Essegaier, S.: Rendezvous search on a graph. Journal of Applied Probability 36, 223–231 (1999)

6. Anderson, E., Weber, R.: The rendezvous problem on discrete locations. Journal of Applied Probability 28, 839–851 (1990)
7. Anderson, E., Fekete, S.: Asymmetric rendezvous on the plane. In: Proc. 14th Annual ACM Symp. on Computational Geometry (1998)
8. Anderson, E., Fekete, S.: Two-dimensional rendezvous search. Operations Res. 49, 107–118 (2001)
9. Arkin, E.M., Mitchell, J.S.B., Piatko, C.D.: Bicriteria shortest path problems in the plane. In: Proc. 3rd Canad. Conf. Comput. Geom., pp. 153–156 (1991)
10. Blum, H.: A transformation for extracting new descriptors of shape. In: Whaten-Dunn, W. (ed.) Proc. Symp. Models for Perception of Speech and Visual Form, pp. 362–380. MIT Press, Cambridge (1967)
11. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the Robots Gathering Problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1181–1196. Springer, Heidelberg (2003)
12. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. Theoretical Computer Science 355, 315–326 (2006)
13. Chin, F., Snoeyink, J., Wang, C.A.: Finding the Medial Axis of a Simple Polygon in Linear Time. Discrete Comput. Geom., 382–391 (1995)
14. Dessmark, A., Fraigniaud, P., Kowalski, D., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica 46, 69–96 (2006)
15. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous oblivious robots with limited visibility. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 247–258. Springer, Heidelberg (2001)
16. Gal, S.: Rendezvous search on the line. Operations Research 47, 974–976 (1999)
17. Hershberger, J., Suri, S.: An Optimal Algorithm for Euclidean Shortest Paths in the Plane. SIAM J. Comput. 28, 2215–2256 (1997)
18. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: gathering of asynchronous oblivious robots on a ring. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 446–462. Springer, Heidelberg (2008)
19. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theoretical Computer Science 390, 27–39 (2008)
20. Kowalski, D., Malinowski, A.: How to meet in anonymous network. Theoretical Computer Science 399, 141–156 (2008)
21. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous in a ring. In: Proc. 23rd International Conference on Distributed Computing Systems (ICDCS 2003), pp. 592–599 (2003)
22. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theoretical Computer Science 384, 222–231 (2007)
23. Preperata, F.P.: The medial axis of a simple polygon. In: Gruska, J. (ed.) MFCS 1977. LNCS, vol. 53, pp. 443–450. Springer, Heidelberg (1977)
24. Schelling, T.: The strategy of conflict. Oxford University Press, Oxford (1960)
25. Stachowiak, G.: Asynchronous Deterministic Rendezvous on the Line. In: Nielsen, M., Kucera, A., Miltersen, P.B., Palamidessi, C., Tuma, P., Valencia, F.D. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 497–508. Springer, Heidelberg (2009)
26. Yu, X., Yung, M.: Agent rendezvous: a dynamic symmetry-breaking problem. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 610–621. Springer, Heidelberg (1996)
27. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 599–608 (2007)

Space-Optimal Rendezvous of Mobile Agents in Asynchronous Trees*

Daisuke Baba[1], Tomoko Izumi[2], Fukuhito Ooshita[1], Hirotsugu Kakugawa[1], and Toshimitsu Masuzawa[1]

[1] Graduate School of Information Science and Technology, Osaka University
{d-baba,f-oosita,kakugawa,masuzawa}@ist.osaka-u.ac.jp
[2] College of Information Science and Engineering, Ritsumeikan University
izumi-t@fc.ritsumei.ac.jp

Abstract. We investigate the relation between the time complexity and the space complexity for the rendezvous problem with k agents in asynchronous tree networks. The rendezvous problem requires that all the agents in the system have to meet at a single node within finite time. First, we consider asymptotically time-optimal algorithms and investigate the minimum memory requirement per agent for asymptotically time-optimal algorithms. We show that there exists a tree with n nodes in which $\Omega(n)$ bits of memory per agent is required to solve the rendezvous problem in $O(n)$ time (asymptotically time-optimal). Then, we present an asymptotically time-optimal rendezvous algorithm. This algorithm can be executed if each agent has $O(n)$ bits of memory. From this lower/upper bound, this algorithm is asymptotically space-optimal on the condition that the time complexity is asymptotically optimal. Finally, we consider asymptotically space-optimal algorithms while allowing slowdown in time required to achieve rendezvous. We present an asymptotically space-optimal algorithm that each agent uses only $O(\log n)$ bits of memory. This algorithm terminates in $O(\Delta n^8)$ time where Δ is the maximum degree of the tree.

1 Introduction

1.1 Background and Motivation

In this paper, we investigate the relation between the time and the memory size for each mobile agent to solve the rendezvous problem. In the problem, agents, which are initially distributed in a network, have to meet at a single node. The rendezvous problem is one of the most fundamental problems as a building block of many agent-based applications. For example, an application may require rendezvous to share the information among all the agents.

Initially agents have no knowledge of the network topology or other agents. This requires agents to move around the network and to determine the meeting

* This work is supported in part by Global COE Program of MEXT, Grant-in-Aid for Scientific Research ((B)19300017, (B)20300012) of JSPS, and the Kayamori Foundation of Informational Science Advancement.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 86–100, 2010.

node based on the collected information. The rendezvous problem can be easily solved if each node in a network has a unique identifier or ID: Each agent explores the network and terminates at the node with the smallest ID. However, such unique ID may not be available to the agents in some cases. It may be prohibited to publish the node ID to agents for security reasons, or the agents cannot store it because of its small memory. Hence, it is important to design algorithms which work in anonymous networks.

In anonymous networks, agents using the same deterministic algorithm cannot meet at a single node in some networks with cycles. The problem can be feasible with some additional assumptions such that agents can leave marks on nodes or the network topology is restricted. In this paper, we investigate the rendezvous problem in acyclic networks or trees while keeping agents from leaving marks.

In the rendezvous problem, the time complexity and the space complexity of agents are important metrics to show the efficiency of algorithms. Because agents are moving entities in the computer network, the size of each agent, which is the memory space of the agent, is desired to be small. In addition, meeting at a single node is not a goal of the agent system but a means to achieve another task. Therefore, the algorithm which achieves rendezvous in a short time is required.

1.2 Related Work

A number of researchers have studied the rendezvous problem in a variety of models. In the settings of anonymous networks and anonymous agents, some rendezvous algorithms allow agents to leave identical tokens on nodes. Kranakis et al. showed that two agents in a synchronous ring network can meet by the same deterministic algorithm using a token per agent [11]. Later, this work is extended to consider any number of agents: If one token is available for each agent, the rendezvous problem in a synchronous ring is solvable for any number of agents [4,6]. The lower bound on the space complexity per agent in this model is $\Omega(\log k + \log\log n)$ where k is the number of agents and n is the number of nodes, and the asymptotically space-optimal algorithm is proposed for unidirectional ring networks by Gasieniec et al. [6]. The effect of token failure is also considered in some papers [1,2,3].

The model described above is different from ours in terms of availability of the memory of nodes. Fraigniaud et al. proved that two anonymous agents can rendezvous in any synchronous tree network without using a token unless the tree is symmetric [5]. This is the same model as we consider in this paper. The memory size of this algorithm is $O(\log n)$ and this is asymptotically optimal [5]. However, if the number of agents is larger than two or the agents move asynchronously, this algorithm does not work.

Lastly, we refer to another model on which agents are oblivious or with no memory but can take the snapshot of the whole system [8,9]. This model is complementary to our model where agents are non-oblivious but can take only the local view of the system.

1.3 Our Contributions

The main goal of this paper is to reveal the relation between the time complexity and the space complexity for the rendezvous problem with k agents in asynchronous tree networks. To achieve the goal, we investigate the following extremes.

1. We consider *asymptotically time-optimal* algorithms and investigate the *minimum memory requirement* per agent for the asymptotically time-optimal algorithms.

We can easily show that $\Omega(n)$ time is required (in the worst case) for agents to meet at a node in trees with n nodes. Then, we show that there exists a tree in which $\Omega(n)$ bits of memory per agent is necessary to solve the rendezvous problem in $O(n)$ time. Last, we present the asymptotically *space-optimal* algorithm on the condition that the time complexity is asymptotically optimal, i.e. both the time complexity and the space complexity are $O(n)$.

2. We consider *asymptotically space-optimal* algorithms while allowing slowdown in time required to achieve rendezvous among agents.

We present an algorithm such that any number of agents with $O(\log n)$ bits of memory, which is asymptotically space-optimal, can rendezvous in any asynchronous tree unless the tree is symmetric. This algorithm attains a significant improvement compared to the previous one [5] in that this algorithm is applicable for any number of agents and any asynchronous non-symmetric tree.

2 Terminology and Preliminaries

2.1 The Network Model

We consider the tree network $T = (V, E)$ with $n = |V|$ nodes where V is the set of anonymous nodes and E is the set of undirected edges. A tree network is an arbitrary connected network with no cycle. Every node u has some ports, each of which connects to an edge incident to the node. The number of ports node u has is denoted by $\delta(u)$ or *degree* of u. We consider that every edge is locally labeled using a *local labeling function* λ_u. That is, each edge e at node u is uniquely labeled by $\lambda_u(e)$ from the set $\{1, \ldots, \delta(u)\}$. We call this label *port number*. This local labeling function or the port number is determined at each node and there is no coherence between $\lambda_u(e)$ and $\lambda_v(e)$ for any $e = \{u, v\} \in E$.

There are $k \geq 2$ agents with no identifiers in the tree T. Each agent has a bounded amount of memory. No agent can use the local memory of any node to leave information on a node. Each agent initially stays at a node called its *home node* and starts the same deterministic algorithm at any time. The agents have no priori knowledge about the network or other agents, that is, they do not know n, k, the shape of the tree, or where other agents are. After the algorithm is started, the agent can move in the network by the following three operations:

1) When the agent walks across an edge e into node v (resp. immediately after the agent initiates the algorithm at node v), it memorizes $\lambda_v(e)$ (resp. 0) and the degree of v. 2) The agent computes internally at v, and determines the port number it leaves next or notices that it should terminate. 3) If the agent decides to move to a neighboring node, it leaves node v through the port which is determined by the previous operation. These actions, such as computing or moving, are progressed asynchronously in the sense that the processing period is finite but there is no assumption of the upper bound on the length of the period. The agent can recognize other agents on the same node, however, it cannot recognize other agents on any edge: Even if two or more agents are passing through the same edge, they cannot detect each other regardless of the direction of each agent.

2.2 Definition of Terms and Problem

The *path* $P(v_0, v_k) = (v_0, v_1, \ldots, v_k)$ with length k is a sequence of nodes from v_0 to v_k such that $\{v_i, v_{i+1}\} \in E$ $(0 \leq i < k)$ and $v_i \neq v_j$ if $i \neq j$. Note that, for any $u, v \in V$, $P(u, v)$ is unique in a tree. The *distance* from u to v, denoted by $dist(u, v)$, is the length of the path from u to v. The *eccentricity* $r(u)$ of node u is the maximum distance from u to an arbitrary node, i.e., $r(u) = \max_{v \in V} dist(u, v)$. The *diameter* D of the network is the maximum eccentricity in the network. The *radius* R of the network is the minimum eccentricity in the network. A node with eccentricity R is called a *center*.

A tree T is *symmetric* iff there exists a function $g : V \to V$ such that all the following conditions hold:

1. For any $v \in V$, $v \neq g(v)$ holds.
2. For any $v \in V$, $\delta(v) = \delta(g(v))$ holds
3. For any $u, v \in V$, u is adjacent to v iff $g(u)$ is adjacent to $g(v)$.
4. For any $\{u, v\} \in E$, $\lambda_u(\{u, v\})$ is equal to $\lambda_{g(u)}(\{g(u), g(v)\})$.

In the *rendezvous problem*, $k \geq 2$ agents have to meet at a single node, which is not predetermined. The node where all the agents meet is called the *rendezvous point*. However, if the tree T is symmetric, there are some cases that agents with the same deterministic algorithm cannot meet at a single node [5]. For this reason, we modify the requirement of the rendezvous problem: All the agents terminate at a common single node if T is not symmetric, otherwise all the agents terminate at two neighboring nodes. We say an algorithm $\mathcal{A}$ *solves* the rendezvous problem if agents executing $\mathcal{A}$ satisfy the above conditions for any tree, any location of home nodes, any starting time of agents, and any execution of agents. The efficiency of an algorithm $\mathcal{A}$ is measured by the *time complexity* and the *space complexity*. The time complexity of $\mathcal{A}$ is defined as the maximum number of movements for an agent because there is no assumption about the period of each action of agents in asynchronous systems. The space complexity of $\mathcal{A}$ is defined as the maximum number of bits required for an agent to store all the local variables.

2.3 Basic Properties

In this subsection, we show basic properties of tree networks. Due to space limitation, we omit the proof of Theorem 2.

Theorem 1 ([10]). *There exist one or two center nodes in a tree. If there exist two center nodes, they are neighbors.*

Theorem 2. *Let v be any node in a tree and v' be the farthest node from node v (i.e., $dist(v, v') = r(v)$). The eccentricity of node v' is equal to the diameter of the tree, that is, $r(v') = D$.*

Theorem 3 ([10]). *Let the distance from node u to node v be D. The node c is a center if and only if c is included in the path $P(u, v)$ and $r(c) = \lceil \frac{D}{2} \rceil$ holds.*

3 Asymptotically Time-Optimal Rendezvous

3.1 Lower Bound on the Memory Space

In this section, we discuss the lower bound on the time complexity and the lower bound of the space complexity of asymptotically time-optimal algorithms for the rendezvous problem. As for the time complexity, Theorem 4 clearly holds.

Theorem 4. *To solve the rendezvous problem in a tree network with n nodes, it takes at least $\Omega(n)$ time to terminate.*

Next, we consider how much memory space per agent is required to solve the rendezvous problem in $O(n)$ time. In fact, the answer is $\Omega(n)$ and we show that there are no algorithms that solve the rendezvous problem in $O(n)$ time with $o(n)$ bits of memory per agent.

Consider an arbitrary algorithm $\mathcal{A}$ which solves the rendezvous problem in $O(n)$ time. For every algorithm $\mathcal{A}$, there exists an execution where no two agents meet until the end of the algorithm. This means that every agent executing $\mathcal{A}$ has to determine the rendezvous point by itself.

In what follows, we consider the case where two agents execute the same deterministic algorithm $\mathcal{A}$ and that the tree T is a line with even number of nodes. We define v_i as the i-th node in the line. Let $L = (V_L, E_L)$ be the subtree of T where $V_L = \bigcup_{1 \leq i \leq \frac{n}{3}} \{v_i\}$ and $E_L = \bigcup_{1 \leq i < \frac{n}{3}} \{v_i, v_{i+1}\}$, $R = (V_R, E_R)$ be the one where $V_R = \bigcup_{\frac{2n}{3}+1 \leq i \leq n} \{v_i\}$ and $E_R = \bigcup_{\frac{2n}{3}+1 < i \leq n} \{v_{i-1}, v_i\}$, and M be the one such that $T \setminus (L \cup R)$ (To understand L, M, and R, see the topology of Figure 1). Let $\lambda_L = \bigcup_{v \in V_L} \lambda_v$ be the labeling for L, $\lambda_M = \bigcup_{v \in V_M} \lambda_v$ for M, and $\lambda_R = \bigcup_{v \in V_V} \lambda_R$ for R. We consider any labeling for λ_L and λ_R, and we consider a certain symmetric labeling α for λ_M (i.e. $\lambda_M = \alpha$). We define $T(\gamma_a, \gamma_b)$ as the tree with $\lambda_L = \gamma_a$ and $\lambda_R = \gamma_b$. Note that, since the topologies of L and R are the same, we can define tree $T(\gamma_a, \gamma_a)$ and $T(\gamma_a, \gamma_a)$ is symmetric.

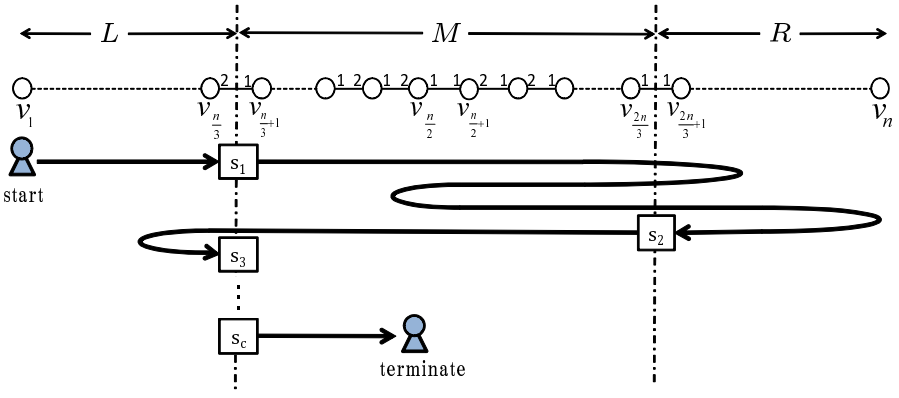


Fig. 1. The numbers found near edges represent port numbers. The bold solid arrow represents the movement of the agent and each s_i represents the i-th migration state of the agent.

In the execution of the algorithm $\mathcal{A}$, the agent traverses the subtree L, M, and R with changing its state (the set of variable value of the agent). We say that the agent *migrates* from L (resp. R) to M if and only if it moves from L (resp. R) to M and it moves from M to R (resp. L) before it moves from M to L (resp. R). If the agent makes its i-th migration with state s, we define s as the *i-th migration state* of the agent. Let $S = s_1 s_2 \dots s_c$ be the *sequence of migration states (SMS)* of the agent if the agent migrates exactly c times in the execution of $\mathcal{A}$ and each s_i denotes the i-th migration state of the agent (See Figure 1). We define $S(\gamma_a, \gamma_b)$ as the SMS of the agent that executes $\mathcal{A}$ with home node v_1 in $T(\gamma_a, \gamma_b)$.

In the followings, we first show that there exist distinct labelings γ_a and γ_b such that $S(\gamma_a, \gamma_a) = S(\gamma_b, \gamma_b)$ holds if each agent has only $o(n)$ bits of memory, and then we show the lower bound of memory space.

Lemma 1. *There exist two distinct labelings γ_a and γ_b such that $S(\gamma_a, \gamma_a) = S(\gamma_b, \gamma_b)$ holds if each agent terminates in $O(n)$ time and has only $o(n)$ bits of memory.*

Proof. Let c' be the maximum number of migrations the agent whose home node is v_1 makes in the execution of $\mathcal{A}$. Note that $c' = O(1)$ holds because $\mathcal{A}$ terminates in $O(n)$ time. Let $S = s_1 s_2 \dots s_c$ be the SMS of the agent where c is lower than or equal to c'. Since there are $\Omega(n)$ nodes in L and in R, there exist $2^{\Omega(n)}$ labelings. However, there are only $2^{c' \cdot o(n)}$ SMSs since the agent has $o(n)$ bits of memory and there exist $2^{o(n)}$ of different states for the agent. Thus, the number of labelings is asymptotically larger than that of SMSs, and there must exist a pair of different labellings γ_a and γ_b such that $S(\gamma_a, \gamma_a) = S(\gamma_b, \gamma_b)$ holds. □

Lemma 2. *Let γ_a and γ_b be distinct labelings such that $S(\gamma_a, \gamma_a) = S(\gamma_b, \gamma_b)$ holds. Then, if algorithm $\mathcal{A}$ solves the rendezvous problem in $T(\gamma_a, \gamma_a)$ and in $T(\gamma_b, \gamma_b)$, it cannot solve the problem in $T(\gamma_a, \gamma_b)$.*

Proof. Let $T_1 = T(\gamma_a, \gamma_a)$, $T_2 = T(\gamma_b, \gamma_b)$, and $T_3 = T(\gamma_a, \gamma_b)$ and we assume that $S(\gamma_a, \gamma_a) = S(\gamma_b, \gamma_b) = s_1 s_2 \dots s_c$ holds.

Consider two agents a_1 and a_2 executing $\mathcal{A}$ in T_1. Let the home node of a_1 be the node v_1 in L and that of a_2 be the node v_n in R. An important observation is that a_1 and a_2 act in a symmetric fashion because T_1 is symmetric. Thus, the i-th migration state of a_2 is also s_i for any $i \leq c$. Since $\mathcal{A}$ solves the rendezvous problem in T_1, the two agents terminate at two symmetric and neighboring nodes in M, which are two center nodes in M.

Next, consider two agents a_1 and a_2 executing $\mathcal{A}$ in T_2. Similarly, we can show that the two agents terminate at the two center nodes in M. We can see from $S(\gamma_a, \gamma_b) = S(\gamma_b, \gamma_b)$ that the terminal node of each agent a_i is the same in T_1 and T_2.

Now, we consider the tree T_3. First, we consider an agent a_1 whose home node is v_1. The agent a_1 in T_3 behaves in the same way as a_1 in L and M of T_1 and it makes the first migration with state s_1 because the labeling of T_3 is the same as that of T_1 for L and M. After the first migration, a_1 behaves in the same way as a_1 in R and M of T_2, and it makes the second migration with state s_2 because the labeling of T_3 is the same as that of T_2 for M and R. By repeating the argument, we can show that the terminal node of a_1 in T_3 is the same as T_1 and T_2. Similarly, we can show that the terminal node of a_2 in T_3 is the same as T_1 and T_2. Recall that the terminal nodes of a_1 and a_2 are not the same but neighboring. However, since T_3 is not symmetric, two agents must terminate at one node. This means agents executing algorithm $\mathcal{A}$ cannot solve the problem in T_3. □

From Lemma 1 and Lemma 2, we can state the following theorem.

Theorem 5. *If an algorithm $\mathcal{A}$ is asymptotically time-optimal (i.e. $O(n)$) for the rendezvous problem in a tree network with n nodes, it requires $\Omega(n)$ bits of memory per agent.*

3.2 The Algorithm with $O(n)$ Memory Space

Outline of the Algorithm. In this subsection, we present the asymptotically time-optimal algorithm Rendezvous-T, which requires $O(n)$ memory space per agent. In Rendezvous-T, each agent traverses the tree, finds a center, and terminates at the center (i.e., the rendezvous point is the center). Note that, since each agent does not have the number k of agents, operates asynchronously, and does not detect other agents passing the same edge, there is an asynchronous execution where no agent meets with another agent until it terminates. Thus, each agent does above works (traverses the tree and finds a center) independently. In the execution of Rendezvous-T, each agent performs seven phases below. Figure 2 shows the locations of the agent according to the execution sequence.

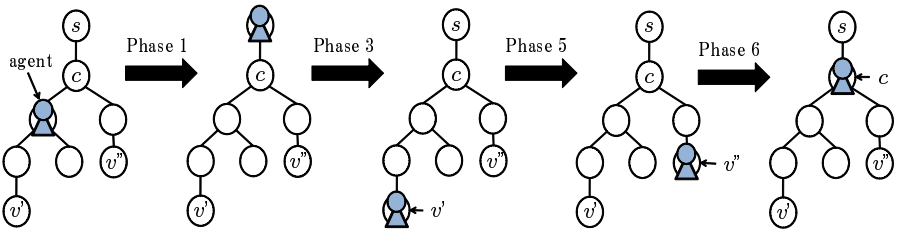


Fig. 2. The leftmost figure shows the initial node of the agent. The other figures show the locations of the agent at the end of Phase 1, Phase 3, Phase 5, and Phase 6.

Phase 1. The agent moves to a leaf node s (we call node s the *first node*).
Phase 2. The agent computes the eccentricity $r(s)$ of the first node s.
Phase 3. The agent moves to the farthest node v' from s (we call node v' the *second node*).
Phase 4. The agent computes $r(v')$ of the second node v' (i.e., $r(v') = D$ from Theorem 2).
Phase 5. The agent moves to the farthest node v'' from v' (we call node v'' the *third node*).
Phase 6. The agent moves to a node c such that c is included in $P(v', v'')$ and $dist(v', c) = \lceil \frac{D}{2} \rceil$ holds. (The node c is one of the centers from Theorem 3.)
Phase 7. If there exist two centers (i.e., D is odd), the agent chooses the rendezvous point from two centers and terminates there.

First, we introduce the *basic step*. The basic step is a traditional technique, which makes each agent traverse the tree in the DFS-traversal. In the basic step, when the agent arrives at node u through the port i, it leaves u through the port $(i+1)$ mod $\delta(u)$ in the next step. The agent starts its basic steps by leaving the port 1. From the definition of our model, the basic steps can be done with $O(\log n)$ bits of memory (the port passed through must be stored). We also define the *reverse step* as the backward step of the basic step. In our algorithms, the agent starts the reverse step only after its basic steps. When an agent starts the reverse step at a node, it leaves the node through the port passed in the previous step.

In the followings, we describe the details of Rendezvous-T. In Phase 1, each agent can move to a leaf using basic steps. It continues its basic steps until it visits a leaf node. A leaf node the agent visits for the first time in Phase 1 is the first node s. In Phases 2 to 4, each agent computes the diameter D. After that, the agent moves to the center in Phases 5 to 6. Note that there may be one or more nodes whose distance from the second node v' is $\lceil \frac{D}{2} \rceil$. To detect the center among them correctly, the agent once moves to the third node v'' whose distance from v' is D because the center node is on the path from v' to v'' (refer Theorem 3). That is, the center node c is the one which the agent, traversing the tree using basic steps, visits for the first time after leaving v'' such that $dist(v', c) = \lceil \frac{D}{2} \rceil$ holds. In Phase 7, the agent terminates at the rendezvous point, which is one of the centers. The agent can also understand whether the tree is symmetric or not in Phase 7.

To realize Rendezvous-T, we introduce two functions MoveAndCompute and Choose. By calling function MoveAndCompute(h_1, h_2) at node v (called *initial node*), the agent starts DFS-traversal of the tree using basic steps. The agent continues the basic steps until it visits node s_2 satisfying $dist(v, s_2) = h_2$ after it visits node s_1 satisfying $dist(v, s_1) = h_1$, and stops the execution of MoveAndCompute at s_2. The agent keeps the maximum distance from the initial node v to a visited node during the execution of MoveAndCompute, and the maximum distance is returned as the output of MoveAndCompute. Function MoveAndCompute is used in Rendezvous-T as follows: In Phase 2, the agent executes MoveAndCompute$(1, 0)$ at a first node s. The agent stops execution of MoveAndCompute$(1, 0)$ when it completes a DFS-traversal. Thus, $r(s)$ can be found as the maximum distance from the first node during the DFS-traversal. In Phase 3, it moves to v' with $dist(s, v') = r(s)$ by calling MoveAndCompute$(0, r(s))$ at s. In Phase 4, MoveAndCompute$(1, 0)$ is executed just like Phase 2. In Phases 5 to 6, by calling MoveAndCompute$(D, \lceil \frac{D}{2} \rceil)$ at the second node v', the agent moves to the third node, and then it moves to a center c. Function Choose chooses a rendezvous point from two centers, and it is used to execute Phase 7 when there exist two centers.

Implementation of MoveAndCompute. In what follows, we explain how to implement MoveAndCompute in Rendezvous-T. In the function MoveAndCompute, the agent keeps the distance from its initial node v to a current node. To compute the distance in anonymous networks, the agent uses the following strategy: Whenever the agent leaves a node u and moves to an adjacent node, it checks whether the next step leads it closer to or farther from its initial node v. The following lemma implies that the agent can decide it by checking the port number the agent will move to. From the property of trees, the lemma clearly holds.

Lemma 3. *Let v be an initial node of* MoveAndCompute. *Assume that when an agent arrived at u for the first time, it passed through the port i at u. When the agent leaves u through the port i', the agent gets closer to v if $i = i'$, and it gets farther from v otherwise.*

By Lemma 3, the agent can calculate the distance from the initial node v to the current node u by computing whether the following condition is satisfied or not: The port through which the agent will pass to leave u is the same as the one through which it visited u for the first time. To compute it correctly, the agent keeps the sequence $H = h_1 h_2 \ldots$ called *history*. The i-th element h_i of the history indicates the i-th movement of the basic steps. Each movement is kept by the fact whether the agent gets closer to the initial node v or farther from v. In more detail, $h_i =$'+' if the agent gets farther from v in the i-th movement, and $h_i =$'−' otherwise. Note that, since each movement is kept with one bit and the agent moves at most $2(n-1)$ times in each MoveAndCompute, the agent requires $O(n)$ memory space to keep the history. By using the history, when the agent leaves a node, it calculates whether it gets closer to the initial node v or farther from v from the following lemma.

Lemma 4. *We assume that an agent visits a node u through the port i' after l basic steps in* MoveAndCompute*, and its history is $H_0 = h_1, h_2, \ldots, h_l$. Let i be the port through which the agent visits u for the first time in the* MoveAndCompute*. Then, the following holds.*

Case1: *If $h_l =$ '+' holds, $i' = i$ holds.*

Case2: *If $h_l =$ '−' holds, we define $H_1, H_2, \ldots$ as follows: Let S_0 be the minimum suffix of H_0 in which the number of '+' is equal to the number of '−'. Then, we define H_1 as the prefix of H_0 such that $H_0 = H_1 S_0$. If the last element of H_1 is '−', we can define H_2 in a similar way. We continue above until the last element of H_t is '+'. Then, $i = (i' - t) \mod \delta(u)$ holds.*

Case 2 in Lemma 3 holds because each suffix S_i corresponds to the DFS-traversal of the subtree with root w where w is one of the children of u. From Lemma 4, when the agent visits u, it can locally compute the port passed when it visited u for the first time. Thus, the agent can determine whether it gets farther from the initial node v or closer to v in the next step locally at u.

We explain the implementation of MoveAndCompute(h_1, h_2) as follows. Let d be the distance from an initial node v to a current node and d_{max} be the maximum number of d the agent has computed. The agent prepares an empty history and sets two variables d and d_{max} to be 0 at v. In MoveAndCompute(h_1, h_2), it performs the following three operations when the agent leaves u: 1) By using the history, the agent determines whether it gets closer to the initial node v or farther from v in the next step. 2) It moves to the neighboring node by the basic step. 3) It updates the history and two values d and d_{max}. The agent can calculate the distance from v to each node it has visited by performing the above three operations repeatedly. If d becomes h_2 after d becomes h_1 once, the agent stops the execution of MoveAndCompute and returns the value of d_{max}.

Implementation of Choose. Here, we explain the implementation of Choose in Rendezvous-T. If the diameter D is even, the algorithm Rendezvous-T is terminated without executing Choose because there is only one center in this case. Thus, we assume that D is odd, i.e., there are two centers in the tree. Let the agent exist at node c, which is one of the centers, after it has completed the execution in Phase 6. Let c' be another center and e be the edge that connects two centers c and c'. When the agent starts the execution of Choose, the agent knows the value of n and D, and it recognizes e in the execution of MoveAndCompute. These values are computed in Phase 2, Phase 5, and Phase 6, respectively. We consider two connected components T_c and $T_{c'}$ by removing edge e from T which include c and c' respectively (See Figure 3).

To choose one of the centers as the rendezvous point, each agent compares T_c with $T_{c'}$. However, if the agent keeps naively the complete map of T_c and $T_{c'}$, it requires $\Omega(n \log n)$ bits. Fortunately, we have a strategy to identify a tree T_c with only $O(n)$ bits. In this strategy, the agent keeps only the degree and the port number of one edge for every node in T_c. The port number kept by the agent on node c is the one connecting to e, and the port number kept by the agent on node $u (u \neq c)$ is the one connecting to the first edge on path $P(u, c)$.

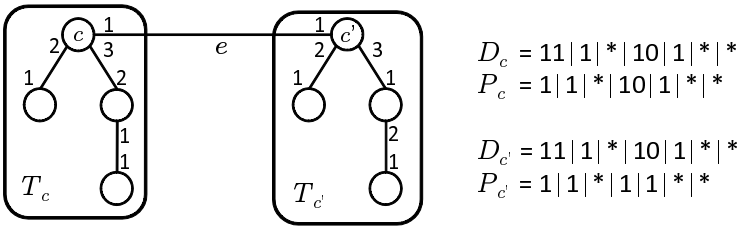


Fig. 3. An example of execution Choose

To realize the above strategy, the agent prepares two sequences D_c and P_c. These sequences are used to keep the degree and the port number for every node in T_c, respectively. Initially, both D_c and P_c are empty words. These sequences are updated whenever the agent makes a movement: When the agent arrives at a node u through an edge e', it appends $\delta(u)$ to D_c and $\lambda_u(e')$ to P_c if it visits u for the first time, and otherwise it appends a symbol $*$. In our algorithm, the sequences D_c and P_c are kept by the agent as a string over alphabet $\Sigma = \{0, 1, |, *\}$. Symbols 0 and 1 are used to represent a degree or a port number as a binary value, and symbol | is inserted between numbers as a separator. For example, sequence $31 * 2$ is represented by $11|1| * |10$. We show an example of D_c and P_c in Figure 3.

The algorithm of Choose consists of three operations: 1) The agent explores the whole of tree T_c using basic steps, and gets the tuple (D_c, P_c). 2) It moves another center c', explores the whole of tree $T_{c'}$, and gets the tuple $(D_{c'}, P_{c'})$. 3) It compares the tuple (D_c, P_c) with $(D_{c'}, P_{c'})$ lexicographically. If these tuples are different, it terminates at the smaller node (c or c'), otherwise it terminates at c' because the tree is symmetric. The following lemma shows the correctness of the algorithm.

Lemma 5. *The tree T is symmetric if both $D_c = D_{c'}$ and $P_c = P_{c'}$ hold.*

Proof. The lemma holds because we can restore an identical subtree from D_c and P_c. Due to space limitation, we omit the proof of the lemma.

Efficiency of the Algorithm. From the proposed algorithm Rendezvous-T, we can state the following theorem.

Theorem 6. *The rendezvous problem in a tree network with n nodes is solved in $O(n)$ time with $O(n)$ memory space on each agent.*

Proof. The time complexity of Rendezvous-T is clearly $O(n)$ because the agent makes at most $2(n-1)$ basic steps in each execution of MoveAndCompute or Choose. Thus, we focus on the space complexity in the followings.

In MoveAndCompute, the agent keeps its history whose length is at most $2(n-1)$ because the number of steps the agent makes is at most $2(n-1)$. Other

variables the agent keeps are clearly at most $O(\log n)$ bits each. Thus, the space complexity of MoveAndCompute is $O(n)$.

Next, we show the space complexity of Choose. To keep all the variables except for the sequences D_c, P_c, $D_{c'}$ and $P_{c'}$, $O(n)$ memory space is sufficient. Thus, we show the number of bits to keep these sequences in the followings.

First we consider the number of bits to keep the sequence D_c. Since the number of nodes included in T_c is lower than n, both of the number of symbol | and that of symbol $*$ included in D_c are at most $n-1$. Let d_c be the total number of symbol 0 and symbol 1 included in D_c. Since symbols 0 and 1 are used in D_c to represent a degree as a binary value for every node u in T_c, the inequality $d_c = \sum_{v \in T_c}(\lfloor \log \delta(v) \rfloor + 1) \leq \sum_{v \in T_c} \delta(v) \leq 2(n-1)$ holds. Thus, the length of D_c is at most $4(n-1)$. Note that since Σ has four symbols, the sequence D_c is stored using at most $8(n-1)$ bits. Next, we consider the number of bits to keep the sequence P_c. By the definition of labeling function λ_v, each port number on node u is lower than or equal to $\delta(u)$ and the number of bits to keep the sequence P_c is not larger than that of D_c. The numbers of bits required for other sequences are shown to be $O(n)$ in the same way and the space complexity of Choose is also $O(n)$. □

4 Asymptotically Space-Optimal Rendezvous

4.1 The Algorithm with $O(\log n)$ Memory Space

In this section, we present an asymptotically space-optimal rendezvous algorithm Rendezvous-S which uses $O(\log n)$ memory space per agent. The idea of the algorithm is the same as that of the previous algorithm Rendezvous-T: The agent moves to a leaf node and then moves to a center by executing MoveAndCompute four times. After that, the agent chooses one of the centers as the rendezvous point by executing Choose if there are two centers. Moreover, the idea of MoveAndCompute is almost the same: Whenever the agent moves to an adjacent node, it computes the distance from the initial node v to the current node u. However, two points are significantly different: 1) One is how the agent determines whether the next step makes it closer to or farther from v in MoveAndCompute, and 2) The other is how the agent chooses one of the centers as the rendezvous point in Choose. These operations require $O(n)$ memory space in Rendezvous-T, however, the agent can use only $O(\log n)$ memory space in this section. Thus, each agent can keep none of the history, the sequence D_c, nor the sequence P_c. We focus on only these two points in the followings.

Before we provide solutions to them, we need to explain two existing functions LogExploration and MatchingEdge, which are proposed to solve the exploration problem for trees [7]. The space complexity of LogExploration is proved to be $O(\log n)$, and the time complexity to be $O(\Delta n^7)$[7], where Δ is the maximum degree of nodes in the network. We use these functions as a subroutine in MoveAndCompute. These functions are proposed in *symmetric-label* trees, where

each edge has the same label in both sides. Thus, in the followings, we consider only symmetric-label trees. Note that, however, this restriction is removed in Section 4.2.

Function LogExploration is the one to realize the exploration in arbitrary trees: The agent traverses all the edges and all the nodes in a tree started at its *home node* v. After the execution of LogExploration, the agent returns back to v. Moreover, the agent can recognize whether the tree is symmetric or not by executing LogExploration. If and only if the tree is symmetric, there exists exactly one edge called *orphan edge* defined by the topology of the tree. In fact, the orphan edge corresponds to an edge connecting two centers in a symmetric tree. Therefore, the agent can check whether the tree is symmetric or not by checking whether there exists an orphan edge. As a subroutine in LogExploration, function MatchingEdge is presented. Let $S = e_1 e_2 \dots e_{2(n-1)}$ be a sequence of edges that an agent passes through when the agent explores the whole tree from node v using basic steps. Since basic steps realize DFS-traversal, each edge is included in S exactly twice. By calling MatchingEdge(i) at v, the agent can compute $j(\neq i)$ satisfying $e_i = e_j$ in S if the tree is not symmetric.

Now, we are ready to settle the first problem: How the agent determines whether the next step makes it closer to or farther from the initial node v in MoveAndCompute? The approach explained below is available only if the tree is not symmetric. However, this never causes a problem: Before the agent starts MoveAndCompute (i.e., before Phase 2 in Section 3.2.1), it executes LogExploration to check whether the tree is symmetric or not. If the tree is symmetric, the agent moves to a node adjacent to the orphan edge and terminates there. Thus, the agent executes the function MoveAndComute only if the tree is not symmetric. Assume the agent visits node u on its $(l-1)$-th basic step. Let $S = e_1 e_2 \dots e_{2(n-1)}$ be the sequence of edges that the agent will pass through in its $2(n-1)$ basic steps from its initial node v. The behavior of the agent consists of the following three operations: 1) The agent returns to v by using $(l-1)$ reverse steps. 2) By calling MatchingEdge(l) at v, it computes j satisfying $e_j = e_l$ in S. If $l < j$ holds, the agent passes e for the first time at the l-th step and thus the l-th step makes the agent farther from v. Otherwise, since the agent has passed e before the l-th step, the l-th step makes the agent closer to v. Note that, only when the agent executes MatchingEdge(l) at v, it can compute whether it passes through e for the first time or not by this way. 3) After the computation, the agent gets back to u by $(l-1)$ basic steps and makes the l-th step.

Next, we explain the implementation of Choose to settle the second problem: How the agent chooses one of the centers as the rendezvous point? When the agent starts Choose, it stays at a center c with recognizing another center c', the edge e connecting c and c', and the value of $f = 2(n-1)$. Besides, the tree T is not symmetric (Otherwise, the agent terminates before it starts MoveAndCompute). Let $t_c[1..j] = p_1, \dots, p_j$ be the sequence of port numbers the agent has left during j basic steps from c. In Choose, the agent compares $t_c[1..f]$ with $t_{c'}[1..f]$ lexicographically and terminates at a center with the smaller one.

The details of Choose in Rendezvous-S are described as follows. Initially, the agent sets the variable i to be 1. The agent makes i basic steps from c and gets the value $t_c[i]$. Next, it returns to c by i reverse steps and moves to c'. Similarly, it gets the value of $t_{c'}[i]$. If $t_c[i]$ is different from $t_{c'}[i]$, then it terminates at the node with the smaller one. Otherwise, the variable i is incremented by one and compares $t_c[i]$ with $t_{c'}[i]$ repeatedly. Since the tree is not symmetric, there is an integer i such that $t_c[i] \neq t_{c'}[i]$.

Since the space complexity of LogExploration is $O(\log n)$ and the time complexity of it is $O(\Delta n^7)$, we can state the following theorem.

Theorem 7. *The rendezvous problem in a tree network with n nodes is solved in $O(\Delta n^8)$ time with $O(\log n)$ memory space on each agent.*

4.2 Extension from Symmetric-Label Trees to General Trees

In the previous subsection, we consider only symmetric-label trees. However, our algorithm works for general trees. The method to obtain a virtual symmetric-label tree T' from an original (general) tree T has been introduced [7]. In this method, a virtual node x is put on every edge $e = \{u, w\}$ such that $\lambda_u(e) \neq \lambda_w(e)$, and port numbers $\lambda_u(e)$ and $\lambda_w(e)$ are assigned to $\lambda_x(\{x, u\})$ and $\lambda_x(\{x, w\})$, respectively. Since virtual tree T' is a symmetric-label tree, the agent can solve the rendezvous problem by executing Rendezvous-S in T'.

We should handle two troubles caused by using the method. First, when the rendezvous point is a virtual node, the agent moves the neighboring real node. That is, if center c is a virtual and the only center, it leaves c through the port 1 and terminates at the arrival node. If there are two centers and one of them is a virtual node, the agent terminates at the real one. Second, when the agent recognizes that virtual tree T' is symmetric and it moves to one of nodes w and w' adjacent to the orphan edge on T', the agent should check whether the real tree T is symmetric or not: Let $v_w(i)$ be *true* (resp. *false*) if the agent in T' visits a real (resp. virtual) node after i basic steps from w. If $v_w(j) = v_{w'}(j)$ holds for all $j < i$ and $v_w(i) \neq v_{w'}(i)$ for some i, the agent can terminate at w (resp. w') if $v_w(i) =$*true* (resp. $v_{w'}(i) =$*true*). If $v_w(i) = v_{w'}(i)$ holds for any integer i ($1 \leq i \leq 2(n' - 1)$, where n' is the number of nodes in T' computed in LogExploration), T is also symmetric and the agent terminates at w without meeting there.

5 Conclusion

In this paper, we have presented two rendezvous algorithms which work with any number of agents in any asynchronous tree. One is asymptotically time-optimal and the other is asymptotically space-optimal. The space complexity of the first algorithm is also asymptotically optimal on the condition that the time complexity is asymptotically optimal.

Our current study does not deal with the solvability in the case that the tree is symmetric but the initial location of agents is asymmetric. It would be an interesting problem to clarify the condition of the initial agent location that makes the rendezvous problem solvable. Our asymptotically space-optimal algorithm takes polynomial but long time to terminate. To construct an asymptotically space-optimal algorithm which solves the rendezvous problem in a shorter time would be also interesting.

References

1. Das, S.: Mobile agent rendezvous in a ring using faulty tokens. In: Rao, S., Chatterjee, M., Jayanti, P., Murthy, C.S.R., Saha, S.K. (eds.) ICDCN 2008. LNCS, vol. 4904, pp. 292–297. Springer, Heidelberg (2008)
2. Das, S., Mihalak, M., Sramek, R., Vicari, E., Widmayer, P.: Rendezvous of mobile agents when tokens fail anytime. In: Proc. 12th International Conference on Principles of Distributed Systems, pp. 463–480 (2008)
3. Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F.L., Santoro, N., Sawchuk, C.: Mobile agents rendezvous when tokens fail. In: Kralovic, R., Sýkora, O. (eds.) SIROCCO 2004. LNCS, vol. 3104, pp. 599–608. Springer, Heidelberg (2004)
4. Flocchini, P., Kranakis, E., Krizanc, D., Sawchuk, C., Santoro, N.: Multiple mobile agents rendezvous in a ring. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 599–608. Springer, Heidelberg (2004)
5. Fraigniaud, P., Pelc, A.: Deterministic rendezvous in trees with little memory. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 242–256. Springer, Heidelberg (2008)
6. Gasieniec, L., Kranakis, E., Krizanc, D., Zhang, X.: Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 282–292. Springer, Heidelberg (2006)
7. Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 585–594 (2007)
8. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: gathering of asynchronous oblivious robots on a ring. In: Proc. 12th International Conference on Principles of Distributed Systems, pp. 446–462 (2006)
9. Klasing, R., Markou, E., Pelc, A.: Gathering asynchrounous oblivious mobile robots in a ring. Theoretical Computer Science 390(1), 27–39 (2008)
10. Korach, E., Rotem, D., Santoro, N.: Distributed algorithms for finding centers and medians in networks. ACM Transactions on Programming Langages and Systems 6(3), 380–401 (1984)
11. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous in a ring. In: Proc. 23rd International Conference on Distributed Computing Systems (ICDCS 2003), pp. 592–599 (2003)

Mobile Robots Gathering Algorithm with Local Weak Multiplicity in Rings

Tomoko Izumi[1], Taisuke Izumi[2], Sayaka Kamei[3], and Fukuhito Ooshita[4]

[1] College of Information Science and Engineering, Ritsumeikan University, Kusatsu, 525-8577 Japan
izumi-t@fc.ritsumei.ac.jp
[2] Graduate School of Engineering, Nagoya Institute of Technology, Nagoya, 466-8555, Japan
t-izumi@nitech.ac.jp
[3] Graduate School of Engineering, Hiroshima University, Higashi-Hiroshima, 739-8527, Japan
s-kamei@se.hiroshima-u.ac.jp
[4] Graduate School of Information Science and Technology, Osaka University, Suita, 565-0871, Japan
f-oosita@ist.osaka-u.ac.jp

Abstract. The gathering problem of anonymous and oblivious mobile robots is one of fundamental problems in the theoretical mobile robotics. We consider the gathering problem in unoriented and anonymous rings, which requires that all robots gather at a non-predefined node. Since the gathering problem cannot be solved without any additional capability to robots, all the previous works assume some capability of robots, such as accessing the memory on node. In this paper, we focus on the multiplicity capability. This paper presents a deterministic gathering algorithm with *local-weak* multiplicity, which provides the robot with the information about whether its current node has more than one robot or not. This assumption is strictly weaker than that by previous works. Moreover, we show that our algorithm is asymptotically time-optimal one, that is, the time complexity of our algorithm is $O(n)$, where n is the number of nodes. Interestingly, in spite of assuming the weaker assumption, it achieves significant improvement compared to the previous algorithm, which takes $O(kn)$ time for k robots.

1 Introduction

1.1 Background and Motivation

Mobile robots are the entities that cooperate with each other by computing, moving and communicating in a plane or in a network. The computational power of mobile robots with quite weak capability is attracting much attention of researchers in the field of distributed computing. In most of the studies, it is assumed that robots are oblivious (no memory to record past situations), anonymous (no IDs to distinguish two robots) and uniform (all robots run the same algorithm). In addition, it is also assumed that each robot has no direct means of communication. Typically, communication among two robots is done in the implicit way that each robot observes the environment, which includes the positions of other robots.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 101–113, 2010.

We consider the gathering problem of mobile robots, which is one of fundamental coordination tasks in the mobile robots system. The problem requires all robots to gather on a non-predefined location. Because of its simplicity, the gathering problem is actively studied in various settings. While most of previous works consider the gathering problem on two-dimensional Euclidean space [1,2,4,7], a number of researches deal with it in graphs [3,5,6,8,9,11,10]. In this paper, we focus on unoriented anonymous rings. That is, we make all robots moving in a graph of ring topology gather on a non-predefined node. Unfortunately, regardless of its settings (graph or 2D-plane), it has been proved that the gathering problem is unsolvable in oblivious and anonymous robot systems without no additional assumption. All of the possibility results depend on some additional assumptions for capability of robots, such as distinct identifiers [10], accessing memory on nodes [3,6,11] or memory on robots [5].

The capability of robots considered in this paper is locality of *multiplicity detection*. The multiplicity detection specifies how each robot observes a node where two or more robots stay. In most of previous works, three types of multiplicity detection are considered: No multiplicity (each robot cannot distinguish the node with a single robot from that with multiple robots), weak multiplicity (each robot can detect whether the number of robots on a node is only one or more than one), and strong multiplicity (each robot can know the number of robots on a node). Recently, in addition to the above types, the notion of *locality* for multiplicity detection capability is introduced [7]. The local multiplicity detection implies that each robot can detect the multiplicity only for its current node. On the other hand, the global multiplicity allows each robot to detect the multiplicity of any node. In the original paper about gathering on ring [9] assumes global-weak multiplicity, and investigate the relationship between its feasibility and initial configurations: It presents a sufficient class of gatherable initial configurations, called *rigid configurations*, which is the set of configurations excluding one having a certain kind of symmetricity. It also proposes a gathering algorithm with global-weak multiplicity for any rigid configuration. However, the strategy of that algorithm strongly depends on the capability of global multiplicity. Its idea is to obtain exactly one node occupied by two or more robots and to make the others reach it. It is clear that this idea does not work correctly if we assume local multiplicity, because any robot on a node cannot detect the multiplicity of other nodes. In this sense, it is an interesting and nontrivial problem to reveal the set of initial configurations for which gathering is possible only with local multiplicity.

1.2 Our Contribution

In this paper, we investigate the feasibility of the gathering with local-weak multiplicity, which is strictly weaker setting than the original works by Klasing et. al. [9]. The contribution of this paper is that any rigid configuration is gatherable only with local-weak multiplicity detection. We propose a deterministic gathering algorithm in rings with k asynchronous robots ($2 < k \leq \lfloor n/2 \rfloor - 1$, where n is the number of nodes) that is applicable to any rigid configuration. Our algorithm assumes that robots are oblivious, anonymous and uniform, and that they have no device to communicate with others directly. To make the gathering problem solvable, we assume that robots have the local-weak multiplicity.

Moreover, the time complexity of our algorithm is also analyzed. In this paper. we evaluate time complexity based on maximum number of *asynchronous rounds* in the worst case, which is one of well-known model for measuring time complexity in asynchronous systems. The time complexity of our gathering algorithm with local-weak multiplicity is asymptotically optimal, that is, $O(n)$. Interestingly, it is significant improvement compared to the previous algorithm, which takes $O(kn)$ rounds [9]: in the previous algorithm, the robots first creates a configuration in which exactly one node v is occupied by two robots. Then, the robots which have no robot on the paths to v move to v, and other robots stay their current nodes. Since it takes $O(n)$ rounds to gather two robots to v, the time complexity of the algorithm is $O(kn)$.

1.3 Road Map

In Section 2 we present the model of autonomous mobile robots considered in this paper, and introduce other necessary notations and definitions. The gathering algorithm with local-weak multiplicity and its time complexity are shown in Section 3. Finally we conclude this paper in Section 4.

2 Preliminaries

2.1 System Models

The system consists of sets of nodes V and mobile robots R. The numbers of nodes and robots are denoted by $n = |V|$ and $k = |R|$ respectively. The nodes construct unoriented and undirected anonymous ring: Neither nodes nor links of the ring have any identifiers and labels. Some nodes of the ring are occupied by robots.

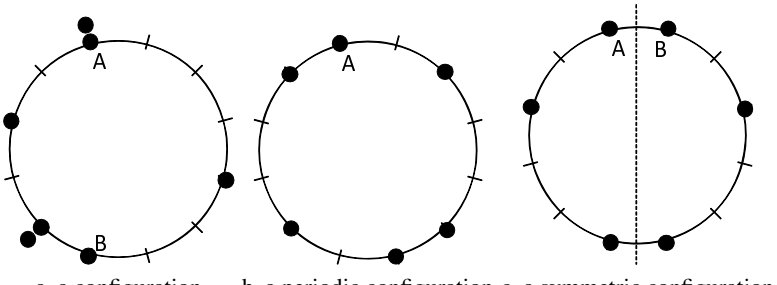
The robots are *anonymous* and *oblivious*. That is, each robot has no identifiers distinguishing itself and others, and cannot explicitly remember the history of its execution. In addition, no device for direct communication is equipped on each robot, such as marks which can be left on nodes and communication devices for sending messages. The cooperation of robots is done in an implicit manner: Each robot observes the configuration (i.e., the nodes occupied by other robots). Each robot executes the same deterministic algorithm in computational cycles (or briefly cycles). At the beginning of a cycle, each robot observes the current configuration and determines to stay idle or to move to one of adjacent nodes based on the algorithm. Then, if the robot decides to move, it moves to the adjacent node. We assume that the robot reaches the destination instantaneously. That is, when a robot observes the configuration, it sees all other robots at nodes and not on links. Cycles are performed asynchronously: The length of time for performing each operation is finite but unbounded. Due to these delay, when a robot moves to an adjacent node which is computed based on the last observation, some robots may stay at different nodes from those observed by the robot.

A configuration is defined by node on which each robot stays. The number of robots staying on a node is called the *multiplicity number* of the node. If the multiplicity number of node v is more than one, we say v is *multiple*. If v has one robot, v is said to be *single*. If there is no robot on a node, the node is called *free*. The *segment* $[v_p, v_q]$ is defined by the sequence $(v_p, v_{p+1}, \ldots, v_q)$ of consecutive nodes in the ring, where

nodes v_p and v_q are single or multiple and the others are free. The distance of segment $[v_p, v_q]$ is equal to the number of nodes in $[v_p, v_q]$ minus 1. Configuration C_i in which $[v_1, v_2], [v_2, v_3] \ldots, [v_w, v_1]$ are consecutive segments in a direction on the ring is defined by a pair of sequences $((d_1^i, d_2^i, \ldots, d_w^i), (m_1^i, m_2^i, \ldots, m_w^i))$, where d_h^i is the distance of $[v_h, v_{(h+1) \bmod w}]$ and m_h^i is the multiplicity number of v_h $(1 \leq h \leq w)$. We get different pairs of sequences when different segment is selected as the first segment or when the distances are listed in the opposite direction in the ring. These sequences represent the same configuration. In this paper, to simplify notation, we use one of the pairs of sequences which represent a configuration to denote the configuration. When no node is multiple in a configuration, we use only sequence $(d_1^i, d_2^i, \ldots, d_w^i)$.

A configuration is changed by movements of robots. Let C_0 be an initial configuration and M_i be a set of movements that occur simultaneously at configuration C_i. An execution is an alternate sequence of configurations and sets of movements $E = C_0, M_0, C_1, M_1, \ldots$, such that occurrence of movements M_i changes the configuration from C_i to C_{i+1}.

Configurations which have no multiple node are classified into three classes in [9]: Configuration C is called *periodic* if C is represented by a concatenation of at least two copies of a subsequence. If there is an axis of symmetry of the ring in configuration C, C is called *symmetric*. The last class is called *rigid*, in which the other configurations are included. Figure 1 shows examples of these configurations.



a. a configuration b. a periodic configuration c. a symmetric configuration

Fig. 1. Examples of configurations: In figure a, the configuration is represented by ((4, 3, 1, 2, 2), (2, 1, 1, 2, 1)) which starts from node A. The periodic configuration in figure b is (2, 3, 1, 2, 3, 1), which is two copies of (2, 3, 1). The configuration in figure c has the axis of symmetry.

By observing a configuration, a robot gets a sequence of distances of segments and information about multiplicity number. Configuration C_i observed by a robot on node v is represented by two sequences of distances $(d_1^i, d_2^i, \ldots, d_w^i)$ and $(d_w^i d_{w-1}^i, \ldots, d_1^i)$ starting from v: One is the sequence in the opposite direction of the other. Since the ring is unoriented, each robot cannot know which is clockwise or counterclockwise one. In this paper, each robot chooses the larger one in lexicographic order: Sequence $(a_i, a_{i+1}, \ldots, a_j)$ is larger than $(b_i, b_{i+1}, \ldots, b_j)$ if there is h $(i \leq h \leq j)$ such that $a_l = b_l$ for $i \leq l \leq h - 1$ and $a_h > b_h$. The capacity of *multiplicity detection* specifies how each robot observes multiple nodes. In this paper, we consider the *local-weak multiplicity*: Each robot can detect whether the multiplicity number of its current node is

more than one. The *view* of a robot on node v at configuration C_i is represented by $s_v(C_i) = (\max\{(d^i_1, d^i_2, \ldots, d^i_w), (d^i_w, d^i_{w-1}, \ldots, d^i_1)\}, m^i)$, where m^i is true if the multiplicity number of node v is more than one and false otherwise. Note that the sequences of distance in views of different robots may be in different direction on the ring. For example, in Figure 1. a., the view of the robots on node A is ((4, 3, 1, 2, 2), true) and that on B is ((3, 4, 2, 2, 1), false). In [9], it is said that a configuration without multiple nodes is rigid if and only if the views of all the robots are different.

2.2 Gathering Problem

The goal of the gathering problem is to gather all the robots on one node that is not predefined and to keep the configuration. For the gathering problem, some impossibility results are shown in [9].

Theorem 1. *The gathering problem is insolvable for the following cases:*

1. *The number of robots is two.*
2. *Robots have no multiplicity detection.*
3. *The initial configuration is any periodic configuration.*
4. *The initial configuration is any symmetric configuration in which axis of symmetry goes through two antipodal links.*

In this paper, we assume that initial configuration is rigid and that $2 < k \leq \lfloor n/2 \rfloor - 1$. That is, in an initial configuration, there is no multiple node and each robot has different view.

The algorithms in this paper are described in some rules. Each rule consists of some conditions and actions of the robots. The robots observe the current configuration and execute actions of one of the rules whose condition matches the observing configuration. Notice that some conditions are described nested if-then-else statements.

3 Gathering Algorithm with Local-Weak Multiplicity

In this section, we present a gathering algorithm with the local-weak multiplicity for any rigid configuration with $2 < k \leq \lfloor n/2 \rfloor - 1$ robots.

Before presenting the details of the algorithm, we introduce several definitions and notation. The *maximum segment* at configuration C_i is the segment with the longest distance in C_i. The *maximum node* at C_i is the non-free node on which the view of robot is the *maximum view* at C_i. If only one node is the maximum one in C_i, the maximum node is denoted by v^i_1 and each non-free node is labeled in the same order as the distance sequence in the view on v^i_1. That is, for view $s_{v^i_1}(C_i) = ((d^i_1, d^i_2, \ldots, d^i_w), m^i)$, node v^i_h is the node such that the distance of segment $[v^i_h, v^i_{h+1}]$ is d^i_h and called h-th node. In this case, for simplicity, distance d^i_h implies the h-th element of the distance sequence on the maximum node.

From the definitions, the following lemma is trivial.

Lemma 1. *Let C_i be a configuration without multiple nodes. The number of the maximum nodes at configuration C_i is one if and only if C_i is rigid.*

In what follows, we present our algorithm in some subsections. In Section 3.1, the gathering algorithm for any rigid configuration C_i where $d^i_1 \geq 4$ and $d^i_2 \geq 3$ is introduced. If a given initial configuration does not match the above conditions, the robots try to create the configuration satisfying them: The algorithm presented in Section 3.2 creates a desired configuration from a rigid one where $d^i_1 \geq 4$ and $d^i_2 < 3$, and in Section 3.3, we present the algorithm for creation of a rigid configuration where $d^i_1 = 4$ from a rigid one where $d^i_1 = 3$. In this paper, we assume that $k \leq \lfloor n/2 \rfloor - 1$ and that initial configuration C_0 is rigid. That is, the distance of the maximum segment is longer than 2 at C_0.

3.1 Gathering from a Rigid Configuration Where $d^i_1 \geq 4$ and $d^i_2 \geq 3$

In this subsection, we assume that the initial configuration C_0 is rigid and satisfies $d^0_1 \geq 4$ and $d^0_2 \geq 3$.

To gather only with the local-weak multiplicity, we must lead a configuration where one node has $k-1$ robots and the other has 1 robot when the number of non-free nodes is 2. The reason is that when two nodes are multiple and the others are free, the robots on the two nodes take the symmetric movements because they have the same view. Hence, when the two nodes are neighboring, all the robots keep staying the current nodes or passing each other. Thus, the robots cannot gather on one node. The idea of our algorithm is that node v^0_2 is kept single, and that the robots on nodes v^0_1 and $v^0_4, \ldots, v^0_w$ gather to node v^0_3. The key to achieve this strategy is that nodes v^0_2 and v^0_3 are kept as the second and the third nodes respectively during the execution.

The details of our algorithm are as follows

- In the case that the number w of non-free nodes is more than or equal to 3,
  - **R1:** When $d^i_w \geq 2$, robots on v^i_1 move to v^i_w.
  - **R2:** When $w \neq 3$, $d^i_w = 1$, $d^i_{w-1} \geq 2$,
    - **R2-1:** if the maximum segment is only $[v^i_1, v^i_2]$ then robots on v^i_w move to v^i_{w-1},
    - **R2-2:** otherwise robots on v^i_2 move to v^i_3.
  - **R3:** When $w \neq 3$ and $d^i_w = 1$ and $d^i_{w-1} = 1$, robots on v^i_1 move to v^i_w.
  - **R4:** When $w = 3$ and $d^i_3 = 1$, robots on v^i_1 move to v^i_3.
- In the case that the number w of non-free nodes is 2,
  - **R5:** robot on the single node moves to the other node occupied by robots.

First, let consider an initial configuration where the number of the maximum segments is one. In our algorithm, the robots move so that the first element on the maximum node is increased, the second element is not changed and the others are decreased. In addition, the last element is kept shorter than the second element. That is, during the execution, the number of the maximum segments is one, and node v^0_2 remains the second node not the first. Figure 2 illustrates an example of executions of the algorithm. The robots on node v^i_1 move to the neighboring node of the last node (rule **R1**). After that, the robots on v^i_w move to the other neighboring node of v^i_w if the neighbor is free (rule **R2-1**). Then, the distance of the last segment becomes 2, and rule **R1** is executed again. That is, the last element of the maximum view is kept shorter than the second element, which is larger than or equal to 3. When d^j_{w-1} becomes 1, the robots on v^j_1 join on v^j_w (rule **R3**). By repeating these actions, the configuration eventually becomes one where $w = 3$ and $d^h_3 = 1$, and then, all the robots except one on the second node gather on v^h_3

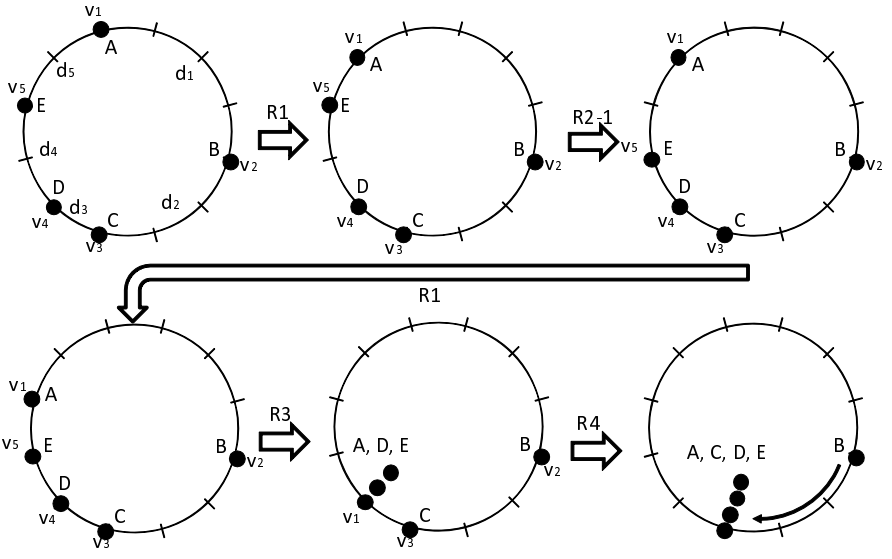


Fig. 2. An example of executions of the gathering algorithm

(rule **R4**). That is, a configuration where one node is single and the other is multiple is created. Lastly, the robot on the single node moves to the multiple node.

We consider the initial configuration C_0 such that more than one the maximum segments exist. In this case, one of rules **R1**, **R2-2** and **R3** is applied. At the next configuration C_1 after executing one of them, the first element on the maximum node v_1^0 is extended by one. That is, the number of the maximum segments becomes one at C_1, and then the robots gather on one node by the above way. However, only when rule **R2-2** is applied, the second element of v_1^0 may become less than 3. It does not satisfy the conditions for execution of this algorithm. In this case, the algorithm presented in Section 3.2, in which the second element becomes 3 keeping only the first segment having the maximum distance, is executed.

Lemma 2. *For $k(2 < k \leq \lfloor n/2 \rfloor - 1)$ robots with the local-weak multiplicity, the algorithm achieves the gathering from a rigid configuration where $d_1^0 \geq 4$ and $d_2^0 \geq 3$.*

Proof. If the configuration becomes one where $w = 3$, $d_3^i = 1$ and the second node is single, it is clear that rules **R4** and **R5** lead to gathering.

In what follows, we explain that the desired configuration is created by rules **R1**-**R3**. The important fact to show this lemma is that the maximum node is only one and the second node is not changed during the execution.

First, we show that in any execution E from a given initial configuration, there is a configuration C_j satisfying the following three conditions: 1) exactly one node v_1^j is the maximum node, 2) only the first segment on v_1^j is the maximum one and 3) $d_2^j \geq 3$. The initial configuration C_0 is rigid. It means that exactly one node is the maximum node (Lemma 1). We consider each case that each rule is applied.

- When rule **R2-1** is applied at C_0, C_0 satisfies the above conditions.
- When rule **R1** or **R3** is executed at C_0, the robot on v_1^0 moves to adjacent node u and the others stay on their current node. Segment $[u, v_2^0]$, whose distance is $d_1^0 + 1$, is only the maximum one because there is no segment whose distance is longer than d_1^0 at C_0. Thus, the first condition is satisfied. The distance of the other segment including u is $d_w^0 - 1$ or 1, and it holds that $d_w^0 \leq d_2^0 = d_2^1$. Thus, node u is only the maximum node and $d_2^1 \geq 3$ at configuration C_1, that is, C_1 satisfies the three conditions.
- When rule **R2-2** is applied at C_0, the robot on v_2^0 moves to adjacent node and the robot on v_1^0 does not move. At the next configuration C_1, since the view on v_1^0 is represented by $(d_1^0 + 1, d_2^0 - 1, \ldots, 1)$, the segment with $d_1^0 + 1$ distance is only the maximum one and v_1^0 is only the maximum node. However, d_2^1 may become less than 3. In this case, the algorithm in Section 3.2 is executed at C_1. Lemma 4 and 5 guarantee that the algorithm leads to a configuration C_j in which $d_2^j = 3$ keeping node v_1^1 and segment $[v_1^1, v_2^1]$ being only the maximum one. That is, configuration C_j satisfies the above three conditions, and rule **R2-2** is not executed at C_j.

Next, we prove that for any configuration $C_i (j \leq i)$ during E, the next configuration C_{i+1} satisfies the following three conditions; 1) only the segment including $[v_1^i, v_2^i]$ has the maximum distance, 2) $v_2^{i+1} = v_2^i$ and 3) $3 \leq d_2^{i+1}$. This implies that the maximum node is only one, node v_2^i remains the second node, and rule **R2-2** is not executed at C_{i+1}. By executing one of rules **R1**, **R2-1** and **R3** at C_i, some of the robots on v_1^i or v_w^i move to adjacent node. We denote the segment including v_1^i and v_2^i at $C_{i+1}$by $[u, v_2^i]$, where u is v_1^i or the neighboring node of v_1^i. Due to the movements, the distance of $[u, v_2^i]$ becomes $d_1^i + 1$ or d_1^i, and each distance of the other segments is decreased, not changed, or kept shorter than 3. Since d_1^i is longer than 3, segment $[u, v_2^i]$ is only the maximum one. Hence, the candidates of the maximum nodes at C_{i+1} are u or v_2^i. Notice that rules **R1**, **R2-1** and **R3** do not change the distance of $[v_2^i, v_3^i]$, and that the other segment neighboring of $[u, v_2^i]$ is shorter than 3 or becomes $d_w^i - 1$. Since $d_w^i \leq d_2^i$ and $3 \leq d_2^i$, the view on v_2^i is smaller than one on u. Thus, node v_2^i remains the second node and $3 \leq d_2^{i+1}$ at C_{i+1}.

At any configuration in the execution until the number of non-free nodes is 2, the second node v_2^j is not changed. From the algorithm, since no robot moves to v_2^j, node v_2^j keeps single. When the last element of view on the maximum node is longer than 2, rule **R1** is applied, and then, the last element eventually becomes 1. After that, rules **R2-1** and **R1** are alternately applied and d_{w-1}^i becomes 1. At this configuration, rule **R3** decrements the number w of nodes occupied by robots. Hence, the configuration becomes eventually one where $w = 3$, $d_3^i = 1$ and the second node is single. □

3.2 Algorithm for a Rigid Configuration Where $d_1^i \geq 4$ and $d_2^i < 3$

In this subsection, we consider the given configuration is rigid and satisfies $d_1^i \geq 4$ and $d_2^i < 3$. The goal of the algorithm presented here is to make a rigid configuration where $d_1^j \geq 4$ and $d_2^j = 3$ from the given configuration.

To present the algorithm, we introduce procedure SHIFT. SHIFT is executed to shift the robots on neighboring nodes in a direction on the ring without creating multiple

nodes. We assume that configuration C_i at which SHIFT is executed satisfies the following conditions:

1. C_i is rigid where $d^i_1 > d^i_2$.
2. There are nodes v^i_p and v^i_q such that $2 \leq d^i_x < d^i_1 (1 < x < p)$, $d^i_y = 1 (p \leq y < q)$ and $d^i_q \geq 2$.

If SHIFT is called then the next rule is executed. By executing SHIFT continuously, the robot on node v^i_p can move away from v^i_{p-1} without creating multiple node.

SHIFT: robot on node v^i_q moves to v^i_{q+1}.

Lemma 3. *Assume that Procedure* SHIFT *is called at a rigid configuration C_i satisfying the two conditions. The configuration C_{i+1} caused by executing* SHIFT *is rigid, and node v^i_1 is also the maximum node at C_{i+1}.*

Proof. The execution of SHIFT does not make any node multiple because the robot on v^i_q gets close to v^i_{q+1} and and $d^i_q \geq 2$.

We show that node v^i_1 is only the maximum node at C_{i+1}. Let u be the node to which the robot on v^i_q moves. By executing SHIFT, the $(q-1)$-th element of the view on v^i_1 is incremented and the q-th element is decremented by one. So, the view on v^i_1 is larger than the previous one. If a node has larger view than or equal to v^i_1 at C_{i+1} then the incremented element must be h-th element of the view on the node ($h \leq q-1$). In addition, except node u, the view which is opposite direction to one on v^i_1 is smaller than the previous one because the decremented element is preceding the incremented element in the view. That is, the other candidates of the maximum nodes are $v^i_2, \ldots, v^i_{q-1}$ and u. For each node $v^i_2, \ldots, v^i_{p-1}$, if the view on the node is the same direction as one on v^i_1 then the conditions on which SHIFT is executed imply that its first element is smaller than d^i_1. Similarly, the first element of the view on each node $v^i_p, \ldots, v^i_{q-2}$ is 1 and one for v^i_{q-1} is 2. For node u, its first element is 2 or $d^i_q - 1$. From the fact that $d^i_1 \geq 3$ and $d^i_1 \geq d^i_q$, these views are smaller than one on v^i_1 at C_{i+1}. Hence, node v^i_1 is only the maximum node at C_{i+1}. From Lemma 1, C_{i+1} is a rigid configuration. □

Now, we explain the algorithm for making a configuration where $d^i_2 = 3$. The algorithm is very simple: Robot on node v^i_3 moves to the neighboring node in order to extend the distance from node v^i_2. If there is a robot on the neighboring node then SHIFT is executed until the robot on v^i_3 can move to the neighbor without creating multiple node. The details of the algorithm are as follows:

– In the case that $d^i_1 \geq 4$ and $d^i_2 < 3$ at configuration C_i,
  R6: When $d^i_3 \geq 2$, robot on node v^i_3 moves away from v^i_2.
  R7: When $d^i_3 = 1$, robot executes SHIFT.

Lemma 4. *Assume that C_i is a rigid configuration where $d^i_1 \geq 4$ and $d^i_2 < 3$. Rules* **R6** *and* **R7** *make a rigid configuration C_j where $d^j_1 \geq 4$ and $d^j_2 = 3$ from C_i. And the maximum node v^i_1 at C_i remains the maximum one.*

Proof. We first show that the next configuration C_{i+1} is rigid and the maximum node v_1^i at C_i remains the maximum one at C_{i+1}. If rule **R7** is applied, Lemma 3 proves this fact. If rule **R6** is applied, the first element of the view on v_1^i is not changed and the second element is incremented. That is, the view on v_1^i becomes larger than the previous one. Let u be the node to which the robot on v_3^i moves. The other candidates of the maximum nodes are nodes whose views contain the incremented element as the first or the second element. In addition, the direction of the candidate's view must be the same direction as one on v_1^i except node u. That is, the other candidates are nodes v_2^i and u. For node v_2^i, if its view is the same direction as one on v_1^i then its first element is incremented but its value is smaller than 4. Similarly, the first element on node u is smaller than 4 or $d_3^i - 1$. Since $d_1^i \geq 4$ and $d_1^i \geq d_3^i$, they cannot be the maximum node. Thus, node v_1^i remains only the maximum node, and C_{i+1} is rigid.

If the configuration satisfies the conditions of rule **R6**, the second element of v_1^i is incremented. Otherwise, rule **R7** is applied until the third element of v_1^i becomes longer than 1, which is a configuration at which rule **R6** is executed. Therefore, the second element of v_1^i eventually becomes 3. □

From the proof of Lemma 4, some segments is incremented but its distance is smaller than 4 during the execution of the algorithm presented in this subsection. Since the distance of the first segment of the maximum node is longer than or equal to 4. Thus, the following lemma is proved.

Lemma 5. *Let C_i be a rigid configuration where the first segment of the view on v_1^i is only the maximum one, $d_1^i \geq 4$ and $d_2^i < 3$. During the execution of rules* **R6** *and* **R7** *until d_2^i becomes 3, the first segment on v_1^i is kept being only the maximum one.*

3.3 Algorithm for a Rigid Configuration Where $d_1^i = 3$

In this subsection, we present the algorithm to make a rigid configuration where $d_1^i = 4$ from the given configuration where $d_1^i = 3$.

The details of the algorithm are as follows:

- In the case that $d_1^i = 3$ at the rigid configuration,
  - **R8:** When $d_w^i \neq 1$, robot on node v_1^i moves to node v_w^i.
  - **R9:** When $(d_2^i, d_w^i) = (1, 1)$, robot executes SHIFT.
  - **R10:** When $(d_2^i, d_w^i) = (2, 1)$,
    - **R10-1:** if configuration $(4, 1, d_3^i, \ldots, d_{w-1}^i, 1)$ is asymmetric then robot on node v_2^i moves to node v_3^i,
    - **R10-2:** else if $d_3^i = 1$ then robot executes SHIFT,
    - **R10-3:** else if configuration $(3, 3, d_3^i - 1, \ldots, d_{w-1}^i, 1)$ is asymmetric then robot on node v_3^i moves to node v_4^i,
    - **R10-4:** otherwise robot on node v_2^i moves to node v_1^i.
  - **R11:** When $(d_2^i, d_w^i) = (3, 1)$, robot on node v_2^i moves to node v_3^i.

In the algorithm, if the robots on node v_2^i or v_w^i can move to the neighboring node without creating a multiple node or a symmetric configuration, the distance d_1^i is extended by the movements of these robots (rule **R8** or **R11**). If both of robots on v_2^i and v_w^i cannot

move, that is, nodes v_2^i and v_w^i are neighbor v_3^i and v_{w-1}^i respectively, then SHIFT is executed to extend distance d_2^i by 2 (rule **R9**). The most sensitive case is that $d_2^i = 2$ and $d_w^i = 1$ because a movement of the robot on v_2^i may lead a symmetric configuration. In the above algorithm, to avoid symmetric configurations, the robots behave different actions in the four cases (rules **10-1** to **10-4**).

Lemma 6. *The algorithm makes a rigid configuration C_i where $d_1^i = 4$ from a rigid configuration where $d_1^i = 3$.*

Proof. We show that when one of the rules is applied at configuration C_i where $d_1^i = 3$, the next configuration C_{i+1} is rigid.

- When the applied rule is **R8** or **R11**, the distance of the segment including v_1^i and v_2^i becomes 4. Since there is no other segment whose distance is 4 at C_{i+1}, the candidates of the maximum nodes are end nodes of the maximum segment. Compared the distances of the neighboring segments of the maximum one, one is smaller than the other. Thus, the maximum node is one of the two end nodes, and C_{i+1} is rigid.
- When rule **R9** or **R10-2** is applied, the next configuration is rigid from Lemma 3.
- When rule **R10-1** is applied, the next configuration $C_{i+1} = (4, 1, d_3^i, \ldots, d_{w-1}^i, 1)$ is asymmetric from the condition for executing **R10-1**. Since the segment whose distance is 4 is only one, the configuration is not periodic. Therefore, C_{i+1} is rigid.
- When rule **R10-3** is applied, it holds that $d_3^i \geq 2$. The next configuration $C_{i+1} = (3, 3, d_3^i - 1, \ldots, d_{w-1}^i, 1)$ has no multiple node and is asymmetric. Since v_1^i is only the maximum node at C_i, there are no other consecutive segments whose distances are 3. That is, C_{i+1} is not periodic. Thus, C_{i+1} is rigid.
- The last case is that rule **R10-4** is executed at C_i. Rule **R10-4** is executed when the configuration does not satisfy the conditions of rules **R10-1** and **R10-3**. It means that configurations $(4, 1, d_3^i, \ldots, d_{w-1}^i, 1)$ and $(3, 3, d_3^i - 1, \ldots, d_{w-1}^i, 1)$ are symmetric, where $d_3^i \geq 2$. In this case, since $(3, 3, d_3^i - 1, \ldots, d_{w-1}^i, 1)$ is symmetric, $d_3^i = 2$. Then, since $(4, 1, 2, \ldots, d_{w-1}^i, 1)$ is symmetric, d_{w-1}^i must be 2. By repeating this way, we know that $d_h^i (3 \leq h \leq w - 1)$ is 2 (see Figure 3). Therefore, configuration C_{i+1} after executing **R10-4** at C_i is $(2, 3, 2, \ldots, 2, 1)$, which is a rigid configuration.

Next, we show that the first element of the maximum view becomes 4. If rule **R8**, **R10-1** or **R11** is applied then it is clear that $d_1^{i+1} = 4$. When rule **R9** is applied and SHIFT is executed repeatedly, the second element eventually becomes 2, in which one of rules **R10** is applied. In the case of rule **R10-2**, by executing SHIFT repeatedly, the configuration becomes one in which **R10-1** is executed, or d_3^j becomes 2 $(j > i)$. In the latter case, some robot moves based on one of rules **R10-1**, **R10-3** and **R10-4**. When rule **R10-3** is applied, rule **R11** is executed at the next configuration C_{i+1}, and when rule **R10-4** is applied, rule **R8** is executed at C_{i+1}. □

From Lemmas 2, 4 and 6, we prove the following theorem.

Theorem 2. *The gathering is achieved from a rigid configuration with $k (2 < k \leq \lfloor n/2 \rfloor - 1)$ robots in the system with the local-weak multiplicity.*

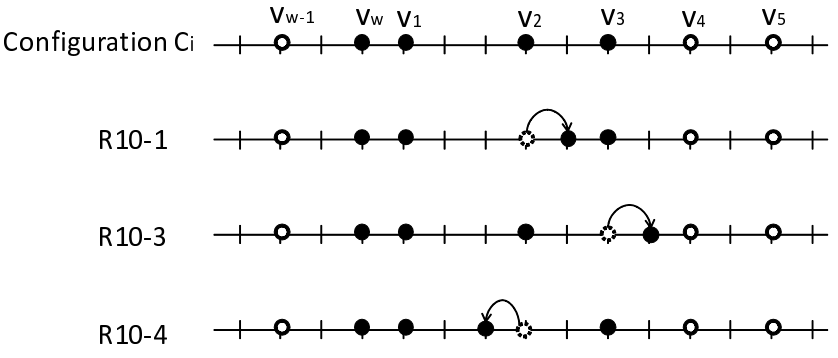


Fig. 3. Each configuration after each execution of **R10-1**, **R10-3** and **R10-4**

3.4 Time Complexity

We show that our algorithm for the gathering with the local-weak multiplicity is $O(n)$-time algorithm in this subsection. In asynchronous systems, time complexity is usually measured in terms of maximum number of *asynchronous rounds* in the worst case. An asynchronous round is defined as the shortest fragment of an execution in which each robot is activated, and moves to the neighboring node or stays the current node at least once.

At procedure SHIFT, at least one robot, which is one on node v_q^i, moves to the neighboring node during one round. That is, it takes $O(h)$ time to extend distance d_p^i from a configuration where $d_p^i, \ldots, d_{p+h}^i = 1$. Since the number of robots is k, value of h is at most k. In each algorithm presented in Section 3.3 and 3.2, consecutive execution of SHIFT occurs at most 2 times. Thus, in the algorithms, it takes $O(k)$ time to execute SHIFT. From the proofs of Lemma 4 and 6, the each algorithm in Section 3.3 and 3.2 leads each desired configuration by applying at most 2 rules except SHIFT. Therefore, the algorithms in Section 3.3 and 3.2 take $O(k)$ time.

In the gathering algorithm in Section 3.1, it takes $O(k)$ time to make a configuration where only one segment is maximum because SHIFT is executed due to rule **R2-2**. After that, the robots on the maximum node take one node toward node v_3^i at two rounds even if the robots is activated at the last of the rounds. Thus, the time complexity to gather all the robots except the robot on v_2^i is $O(n)$. Therefore, the following theorem is proved.

Theorem 3. *The algorithm achieves the gathering in $O(n)$ time.*

4 Conclusions

This paper presented the deterministic gathering algorithm in asynchronous rings with oblivious and anonymous robots using the local-weak multiplicity. The time complexity of our algorithm is $O(n)$, while that of the algorithms with global-weak multiplicity in the previous works are $O(kn)$. This result implies the our algorithm is asymptotically time-optimal one.

In this paper, we restrict the number of robots so that $2 < k \leq \lfloor n/2 \rfloor - 1$. When there are more than $\lfloor n/2 \rfloor - 1$ robots on a ring, the distance between every consecutive robots may be less than or equal to 2 at a initial configuration. In this case, it is hard that each robot moves to adjacent node without creating a multiple node or a symmetric configuration. Our feature work is to present the algorithm or to prove the impossibility of the gathering in this situation. Moreover, the gathering with the local-weak multiplicity from a symmetric configuration is also one of interesting problems.

Acknowledgment. This work is supported in part by The Telecommunication Advancement Foundation and Grant-in-Aid for Young Scientists ((B)19700075) of JSPS.

References

1. Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for automonous mobile robots. SIAM Journal of Computing 36(1), 56–82 (2006)
2. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the robots gathering problem. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003), pp. 1181–1196 (2003)
3. Flocchini, P., Kranakis, E., Krizanc, D., Sawchuk, C., Santoro, N.: Multiple mobile agents rendezvous in a ring. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 599–608. Springer, Heidelberg (2004)
4. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchnorous robots with limited visibility. Theoretical Computer Science 337(1-3), 147–168 (2005)
5. Fraigniaud, P., Pelec, A.: Deterministic rendezvous in trees with little memory. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 242–256. Springer, Heidelberg (2008)
6. Gasieniec, L., Kranakis, E., Krizanc, D., Zhang, X.: Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 282–292. Springer, Heidelberg (2006)
7. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Randomized gathering of mobile robots with local-multiplicity detection. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 384–398. Springer, Heidelberg (2009)
8. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetrics: Gathering of asynchronous oblivious robots on a ring. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 446–462. Springer, Heidelberg (2008)
9. Klasing, R., Markou, E., Pelc, A.: Gathering asynchnorous pblivious mobile robots in a ring. Theoretical Computer Science 390(1), 27–39 (2008)
10. Kowalski, D., Pelec, A.: Polynomial deterministic rendezvous in arbtrary graphs. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 644–656. Springer, Heidelberg (2004)
11. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous in a ring. In: Proceedings of the 23th International Conference on Distributed Computing Systems (ICDCS 2003), pp. 592–599 (2003)

Average Long-Lived Memoryless Consensus: The Three-Value Case*

Ivan Rapaport[1] and Eric Rémila[2]

[1] DIM-CMM (UMI 2807 CNRS), Universidad de Chile,
2120 Blanco Encalada, Santiago - Chile
rapaport@dim.uchile.cl
[2] Université de Lyon, LIP (UMR 5668 CNRS-ENS, Université Lyon 1),
46 allée d'Italie 69364 Lyon Cedex 7 - France
eric.remila@ens-lyon.fr

Abstract. We study strategies that minimize the *instability* of a fault-tolerant consensus system. More precisely, we find the strategy than minimizes the number of output changes over a random walk sequence of input vectors (where each component of the vector corresponds to a particular sensor reading). We analyze the case where each sensor can read three possible inputs. The proof of this result appears to be much more complex than the proof of the binary case (previous work). In the binary case the problem can be reduced to a minimal cut in a graph. We succeed in three dimensions by using the fact that an auxiliary graph (projected graph) is planar. For four and higher dimensions this auxiliary graph is not planar anymore and the problem remains open.

1 Introduction

There are situations where, for fault-tolerant purposes, a number of sensors are placed in the same location. Ideally, in such cases, all sensor readings should be equal. But this is not always the case; discrepancies may arise due to differences in sensor readings or to malfunction of some sensors. Thus, the system must implement some form of fault-tolerant averaging *consensus function* ϕ that returns a representative *output value* of the sensor readings.

Let us consider n sensors which are sampled at synchronous rounds. In each round an *input vector* x of sensor readings is produced, where x_i is a value from some finite set S produced by the i-th sensor. Assuming that at least $t+1$ entries of the vector are correct, ϕ is required to return a value that appears in at least $t+1$ entries of x.

The sampling interval is assumed to be short enough in order to guarantee the sequence of input vectors to be *smooth*: exactly one entry of a vector changes from one round to the next one.

* Partially supported by Programs Fondecyt 1090156, Basal-CMM, Instituto Milenio ICDB, Ecos C09E04 and IXXI (Complex System Institute, Lyon).

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 114–126, 2010.

A natural function ϕ is the one that returns the most common value of vector x. However, the *instability* of such function is high. More precisely, the output value computed by such ϕ could change from one round to the next one unnecessarily often.

For tackling this stability issue a worst case complexity measure was introduced in two previous papers [6,8]. The input sequence considered in those papers was assumed to be, in addition to smooth, *geodesic*: the i-th entry of the input vector was allowed to change *at most once* over the sequence. The instability of a consensus function was given by the largest number of output changes over any such sequence, called a *geodesic path*.

In [1] we introduced, as an alternative measure, a more natural (and subtle) notion called *average instability*. We removed the geodesic requirement and therefore the smooth sequences of input vectors we considered were random walks over the hypercube. If $P = x_0, x_1, \ldots$ is such a walk, then the average instability of a consensus function ϕ is given by the fraction of time ϕ changes its output over P.

We studied in [1] the case when the input is binary ($S = \{0, 1\}$). In particular, for the memoryless case, we proved that function ϕ_0, that outputs 1 unless it is forced by the fault-tolerance requirement to output 0 (on vectors with t or less 1's), is optimal.

In the present paper we analyze the 3-value case ($S = \{0, 1, 2\}$). We extend previous result proving that the natural extension of ϕ_0 is in fact optimal among all the anonymous strategies (a strategy is anonymous, or symmetric, when it only depends on the number of entries of each type, and not in the respective places of the entries).

More precisely, let $\#_b(x)$ be the number of entries of x that are equal to $b \in \{0, 1, 2\}$. Let ψ be the consensus function defined as follows:

- $\psi(x) = 2$ if $\#_2(x) \geq t + 1$,
- $\psi(x) = 1$ if $\#_2(x) \leq t$ and $\#_1(x) \geq t + 1$,
- $\psi(x) = 0$ if $\#_2(x) \leq t$ and $\#_1(x) \leq t$.

In this paper we prove that the consensus function ψ has optimal stability according to the average criterion (among all the consensus symmetric memoryless functions).

The proof of this result is much more complex than the proof of the binary case. In the binary case the problem can be reduced to a minimal cut in a graph. For higher dimensions this approach can not be used. In fact, the problem becomes a multiterminal cut problem, which is NP-hard [5].

We succeed in 3 dimensions by using the fact that an auxiliary graph (projected graph) is planar. Unfortunately, the auxiliary graph is not planar in 4 (and higher) dimensions. We are therefore facing a hard combinatorial problem. In fact, the multiterminal cut problem is polynomial for planar graphs and NP-complete in the general case.

We would like to point out that Davidovitch et al [6] faced the same change in the difficulty level when they extended their binary case result to a more

general, multi-valued result. In fact, for that extension, they had to use topological techniques in high dimensional complexes.

As noted in [8], studying the instability of consensus functions may have applications in various areas of distributed computing, such as self-stabilization [7] (indeed, see [11]), Byzantine agreement [2], real-time systems [12], complexity theory [10] (boolean functions), and VLSI energy saving [3,14,17] (minimizing the number of transitions).

2 The Model

Let n, t, k be non-negative integers such that $n \geq kt+1$. The set of *input vectors* is the set $V = \{0, 1, ..., k-1\}^n$. The *input space* is the graph (V, E) where the edges E are all the (unordered) pairs of vectors which differ in exactly one component. Notice that $|E| = n(k-1)k^n/2$. In this paper we focus in the case $k = 3$.

The *distance* $d(x_1, x_2)$ between two vectors x_1, x_2 is the distance in the input space, i.e., the number of entries in which x_1 and x_2 differ. We denote by $\#_b(x)$ the number of entries of x that are equal to $b \in \{0, 1, 2\}$. The *corners* of the input space are the vectors 0^n, 1^n and 2^n.

Let $x_0, x_1, x_2, \ldots$ be a *walk* in the input space, i.e., a sequence $(x_i)_{i\in\mathbb{N}}$ of vectors in V such that for each $i \in \mathbb{N}$, the pair $\{x_i, x_{i+1}\}$ is an element of E.

A consensus decision function ϕ assigns, to each x_i, an *output value* $d \in \{0, 1, 2\}$. In this paper we only consider such *memoryless* functions (for which the decision depends only on the current input vector x_i, and not on the past history).

Formally, $\phi : V \to \{0, 1, 2\}$ is the *consensus decision function*. The fault-tolerance requirement that ϕ must satisfy is the following: $\phi(x) = d \Rightarrow \#_d(x) \geq t + 1$.

A consensus decision function ϕ is *symmetric* if $\phi(x)$ depends only on the three values $\#_0(x)$, $\#_1(x)$,$\#_2(x)$ (which correspond to the number of 0s, 1s and 2s of x).

An *execution* of the system is a sequence $(x_0, d_0) \to (x_1, d_1) \to \ldots$, where $d_i = \phi(x_i)$.

In the sequel we will often refer to the "consensus function" instead of the "consensus symmetric memoryless decision function".

2.1 Stability: Geodesic Criterion

Dolev and Rajsbaum proposed in [8] a criterion for measuring the instability of a consensus function. This criterion is based on a *geodesic execution*, which is an execution where at most one transition is performed in each sensor (we say that the transition $(x_i, d_i) \to (x_{i+1}, d_{i+1})$ is performed in sensor j if j is the index of the component on which x_i and x_{i+1} differ). Notice that a geodesic execution is of length at most n. The *geodesic instability* is the maximal number of changes in a geodesic execution.

In [1] we explained why this criterion is not completely satisfying. The following result about memoryless symmetric systems with three input values gives a new argument in the same line.

Proposition 1. *Let $4t - 1 \geq n$. Then, for any (symmetric memoryless) consensus function ϕ with three values, the geodesic instability of ϕ is equal to n.*

The proof, which uses Sperner's Lemma [4,16], is omitted in this conference version.

2.2 Stability: Random Walk Criterion

In [1] we proposed a new criterion that uses random walks for determining the instability of a consensus function. The idea is the following. The initial input vector x_0 is chosen according to some distribution λ. On the other hand, if x_i is the current input vector, then the next input vector x_{i+1} is chosen in a random uniform way among the vectors at distance one from x_i.

Formally, we have a Markov process (P, μ_0) whose set of states is V and there is a transition from x to x' if $\{x, x'\} \in E$. The probability of such transition is $1/2n$ (this defines the transition matrix P). The initial distribution is $\mu_0 = \lambda$. Each state x_i has an associated output value $d_i = \phi(x_i)$. Therefore the random walk $x_0, x_1, x_2, \ldots$ defines an execution.

We use the classical Kronecker notation $\delta(i, j)$ where $\delta(i, j) = 1$ when $i = j$, and $\delta(i, j) = 0$ otherwise. Let $c_{\lambda,l}(\phi)$ be the random variable defined by: $c_{\lambda,l}(\phi) = \frac{1}{l} \sum_{k=0}^{l-1} \delta(d_k, d_{k+1})$. The *average instability* of a consensus function ϕ is defined by:

$$inst(\phi) = \mathbb{E}(\lim_{l \to \infty} c_{\lambda,l}(\phi)).$$

The average instability represents the frequency of decision changes along a random execution (and, as an easy consequence of the classical Ergodic Theorem [15], it is well defined).

Furthermore, the stationary distribution π of the random walk $x_0, x_1, x_2, \ldots$ is the uniform distribution, i.e., $\pi_x = 1/3^n$ for every $x \in V$ (for notation see [15])).

Since the instability of ϕ counts the number of times the function ϕ changes its decision along a random walk, by the Ergodic Theorem this value tends to the number of bicolored edges (where changes in the decision take place) divided by $|E| = n3^n$. In other words,

$$inst(\phi) = \frac{\sum_{\{x,y\} \in E} \delta_\phi(\{x, y\})}{n3^n} = \frac{|E_\phi|}{n3^n},$$

where $\delta_\phi(\{x, y\}) = 1$ when $\phi(x) \neq \phi(y)$ and $\delta_\phi(\{x, y\}) = 0$ otherwise, and E_ϕ denotes the set of edges e such that $\delta_\phi(e) = 1$ (i.e., E_ϕ is the set of bicolored edges). A detailed proof of the equality above was given for the binary case in [1]. The proof of the 3-value case is identical.

3 Basics for the Symmetric Case

We study here symmetric systems. For these systems it is convenient to use the *projected input space* (V', E', w), which is a weighted graph.

Each vertex v of the projected input space V' can be seen as a triple (i, j, k) with i, j, k nonnegative, and $i + j + k = n$. Each of these triplets represent the set of $\frac{n!}{i!j!k!}$ input vectors containing i 0-entries, j 1-entries and k 2-entries. We say that i, j and k are, respectively the *0-component*, the *1-component* and the *2-component* of v.

On the other hand, two distinct vertices (i, j, k) and (i', j', k') are linked by an edge of E' when $|i - i'| \leq 1$, $|j - j'| \leq 1$ and $|k - k'| \leq 1$ (remark that we necessarily have $i = i'$ or $j = j'$ or $k = k'$).

The weight $w(\{v, v'\})$ of the edge $\{v, v'\}$ is the number of edges $\{x, x'\}$ of E such that v is the projection of x and v' is the projection of x'. For $i < n$ and $j > 0$, the weight of the edge $\{(i, j, k), (i + 1, j - 1, k)\}$ is given by the equality:

$$w(\{(i, j, k), (i + 1, j - 1, k)\}) = \frac{n!}{i!(j - 1)!k!} \tag{1}$$

For each symmetric function ϕ, we define $\phi(v) = \phi(x)$ where x is any input vector such that v is the projection of x. Therefore, in the symmetric case, the last equality of the previous section becomes:

$$inst(\phi) = \frac{\sum_{\{v,v'\} \in E'} w(\{v, v'\})\delta_\phi(\{v, v'\})}{n3^n} = \frac{\sum_{e' \in E'_\phi} w(e')}{n3^n},$$

where $\delta_\phi(\{v, v'\}) = 1$ when $\phi(v) \neq \phi(v')$, $\delta_\phi(\{v, v'\}) = 0$ otherwise, and E'_ϕ denotes the set of edges e' of E' such that $\delta_\phi(e') = 1$ (in other words, E'_ϕ is the set of bicolored edges in the projected input graph). It follows that, when we want to minimize the instability, we have to minimize the quantity $\sum_{e' \in E'_\phi} w(e')$.

The structure of the projected input space is easily understandable. It can be seen as a part of the triangular lattice delimited by an equilateral triangle of side length n, whose extreme points are the corners $(n, 0, 0)$, $(0, n, 0)$ and $(0, 0, n)$ (see Figure 1). In particular, this graph is planar. More precisely, we will use the following drawing: we take three points a, b, c, of the plane $\mathbb{R}^2$, usually $a = (0, 0), b = (n, 0)$ and $c = (\frac{n}{2}, \frac{n\sqrt{3}}{2})$. Each vertex (i, j, k) is identified with the point p_{ijk} which is the barycenter of (a, i), (b, j), and (c, k). Edges are classically represented by lines segments linking neighbor vertices. This representation is called the *canonical representation* of the projected input space.

We associate vectors to edges and faces: the vector corresponding to the edge $e = \{v, v'\}$ is the average between its endpoints vectors, i.e., the vector corresponding to the center of the line segment $[v, v']$. Hence, the vector of the edge $e = \{v, v'\}$ is formed by two semi-integer values and one integer value. The vector of a (finite) triangular face ϕ is the average between its vertices, i.e., the vector corresponding to the center of f. This vector is of the form (x, y, z), with $3x$, $3y$ and $3z$ being all integers (moreover, one can check that $3x$, $3y$ and $3z$ are equal modulo 3).

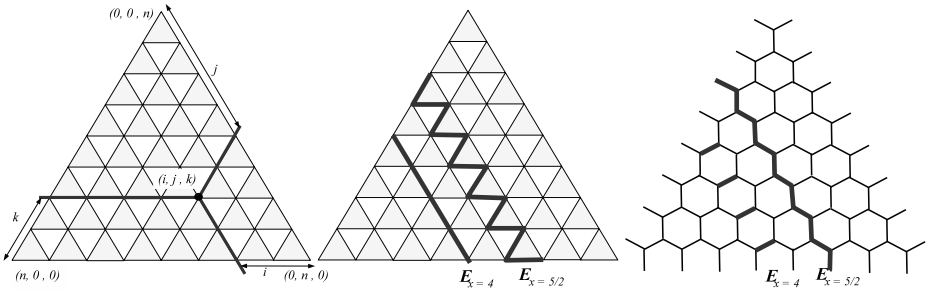


Fig. 1. Conventional notations. Left: triple (i, j, k) encoding a vertex. Center: the sets $E_{x=\frac{5}{2}}$ and $E_{x=4}$. Right: the same sets in the dual graph.

We recall that the dual graph of the projected input space (V', E') is the (multi)graph (F, E') such that F is the set of faces induced by (V', E'), and a pair $\{f, f'\}$ is an edge if there exists $\{v, v'\} \in E'$ such that the line segment $[v, v']$ is shared by both f and f'. As it is usually done, we refer indistinctly to the edge $\{f, f'\}$ of the dual graph and to the same edge $\{v, v'\}$ of the projected input space. Since the canonical representation of the projected input space is part of the triangular lattice of the plane, the canonical representation of the dual graph is part of the hexagonal lattice, with all pending edges linked to a particular vertex of the dual graph, which is the infinite face f_∞.

Let i be an integer. We define the edge set $E_{x=\frac{i}{2}}$ as the set of edges whose 0-component is $\frac{i}{2}$. In other words, $E_{x=\frac{i}{2}}$ is the set of edges intersecting the closed line segment $[(\frac{i}{2}, n-\frac{i}{2}, 0), (\frac{i}{2}, 0, n-\frac{i}{2})]$. Notice that an edge intersects this closed line segment if and only if the corresponding edge of the dual graph also intersects the same line segment (with the convention that the vertices of the dual graph are placed in the center of the triangular faces). One can define, in a similar way, for any integer j, $E_{y=\frac{j}{2}}$ and, for any integer k, $E_{z=\frac{k}{2}}$ (using respectively the 1-component and the 2-component).

Lemma 1. *Let e be an edge of $E_{x=\frac{i}{2}}$ with vector $(\frac{i}{2}, \frac{j}{2}, \frac{2n-i-j}{2})$.*

- *Let $sym(e)$ be the edge with vector $(\frac{i}{2}, \frac{2n-i-j}{2}, \frac{j}{2})$. We have $w(e) = w(sym(e))$.*
- *For i odd, and $j+1 \leq \frac{2n-i}{2}$, let e_+ denote the edge with vector $(\frac{i}{2}, \frac{j+1}{2}, \frac{2n-i-j-1}{2})$*
  - *If j is even, then $w(e) = w(e_+)$,*
  - *If j is odd, then $w(e) < w(e_+)$.*
- *For i even (which enforces j being odd) and $j+2 \leq \frac{2n-i}{2}$, let e_+ denote the edge with vector $(\frac{i}{2}, \frac{j+2}{2}, \frac{2n-i-j-2}{2})$. We have $w(e) < w(e_+)$.*

Proof. This lemma is a direct consequence of equality 1. □

The lemma above describes the evolution of weights of edges in $E_{x=\frac{i}{2}}$. First, it says that there is a weight symmetry with respect to the median axis, formed by

points (x, y, z) such that $y = z$. Secondly, the lemma also says that the weight increases as you move towards the median axis.

Each consensus function ϕ divides the vertices of the projected input space (and therefore the plane since the graph is planar) into three *zones*, one for each output value. Each zone contains the corresponding corner. Notice that zones are not necessarily connected.

Consider the set E_ϕ of bicolored edges induced by ϕ or, more precisely, consider the corresponding edges in the dual graph. These edges form cycles, which surround connected components of the zones.

We call *network* a subset of edges. Given a network N, the weight $w^N_{x=\frac{i}{2}}$ is defined by $w^N_{x=\frac{i}{2}} = min\{w(e) \,|\, e \in E_{x=\frac{i}{2}} \cap N\}$ (we say that $w^N_{x=\frac{i}{2}} = \infty$ when $E_{x=\frac{i}{2}} \cap N$ is empty). This weight represents the minimal necessary weight for passing from one side of the segment $[(\frac{i}{2}, n-\frac{i}{2}, 0), (\frac{i}{2}, 0, n-\frac{i}{2})$ to the other part still remaining in the network N.

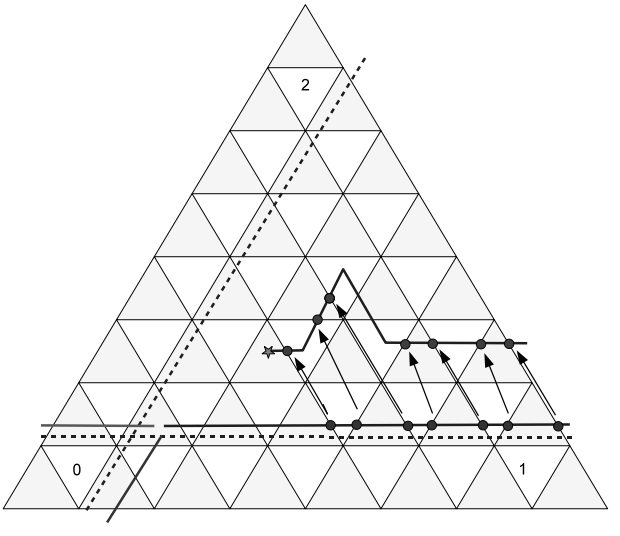


Fig. 2. An application of Lemma 2, with the network N_0 formed by edges whose 1-components and 2-components are both at least 3/2. The weight of the path issued from the star is larger than the weight of the marked edges in the horizontal path issued from the cell of vector $(n - 2t - \frac{2}{3}, t + \frac{1}{3}, t + \frac{1}{3})$ (notice that this path is a boundary path induced by the consensus function ψ).

Lemma 2. *Consider a simple path p (of the dual graph) linking a finite face f_0 to the infinite face f_∞, remaining in a network N, whose last edge belongs to $E_{x=0}$. Let (i_0, j_0, k_0) be the vector corresponding to f_0. The sum $\sum_{e\in p} w(e)$ is denoted by $w(p)$. We have the inequality:*

$$w(p) \geq \sum_{0 \leq i < 2i_0} w^N_{x=\frac{i}{2}}$$

Proof. This is obvious since, for each integer i such that $0 \leq i < 2i_0$, the path must contain an edge of $E_{x=\frac{i}{2}}$ and the sets $E_{x=\frac{i}{2}}$ are pairwise disjoint. □

4 Our Result

The following theorem is the main result of the paper.

Theorem 1. *Consider the consensus function ψ defined by:*

- $\psi(x) = 2$ *if* $\#_2(x) \geq t+1$,
- $\psi(x) = 1$ *if* $\#_2(x) \leq t$ *and* $\#_1(x) \geq t+1$,
- $\psi(x) = 0$ *if* $\#_2(x) \leq t$ *and* $\#_1(x) \leq t$.

The consensus function ψ has optimal stability according to the average criterion, among all symmetric functions.

We decompose the proof into two lemmas from which the theorem is a direct consequence. We first limit ourselves to the case when each of the three zones formed from vertices with the same output value is connected in the merged input graph.

Lemma 3. *Let ϕ be a consensus function for which each of the three zones induced by ϕ is connected. We have $inst(\phi) \geq inst(\psi)$.*

Proof. In this case, E'_ϕ is just formed (in the dual graph) by three edge disjoint paths linking a face $f_0 = (i_0, j_0, k_0)$, with $i_0 > t, j_0 > t$, and $k_0 > t$ to the infinite face f_∞. We call p_0 the path linking f_0 to f_∞ and crossing the set $E_{x=0}$. This path is the boundary between the 1-zone and the 2-zone.

We denote by N_0 the network formed by edges whose vector (x, y, z) are such that $y > t$ and $z > t$. The path p_0 remains in N_0 thus, applying Lemma 2, we get:

$$w(p_0) \geq \sum_{0 \leq i < 2i_0} w^{N_0}_{x=\frac{i}{2}}.$$

In the same way we get $w(p_1) \geq \sum_{0 \leq j < 2j_0} w^{N_1}_{y=\frac{j}{2}}$ and $w(p_2) \geq \sum_{0 \leq k < 2k_0} w^{N_2}_{z=\frac{k}{2}}$.

Adding these inequalities, we get a lower bound for $w(E'_\phi) = w(p_0) + w(p_1) + w(p_2)$. But this bound is not sufficient to get our result. We need a refinement of Lemma 2 using other separating edge sets.

Up to symmetry, it can be assumed that $i_0 \geq j_0$. We define the integer j_1 as the lowest integer such that $j_1 \geq 2j_0$. For $2t < k < 2k_0$, we state $i_k = 2n - k - j_1$ in such a way that $L_k = (i_k/2, j_1/2, k/2)$ and $R_k = (j_1/2, i_k/2, k/2)$ have semi-integer or integer coordinates.

The set $E^{f_0}_{z=k/2}$ is formed by edges listed below (see Figure 3):

- the edges of $E_{z=k/2}$ whose whose 1-component is at least j_1 and whose 0-component is at least j_1 (i.e. the edges which meet the line segment $[L_k, R_k]$).

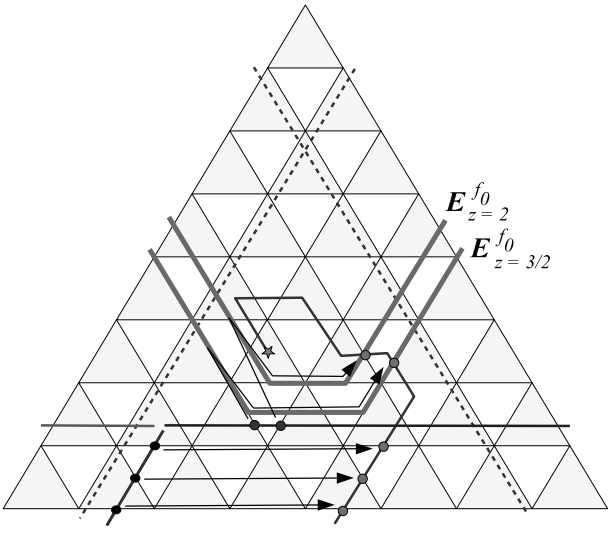


Fig. 3. The improvement of the argument of Lemma 2. We use some "broken lines". For each marked edge of E'_ψ, one can find an edge of larger weight on the path issued from the star.

- the edges of $E_{x=i_k/2}$ whose 2-component is at most k (i.e., the edges which intersect the line segment $[L_k, (i_k/2, 0, n - i_k/2)]$).
- the edges of $E_{j=i_k/2}$ whose 2-component is at most k (i.e., the edges which intersect the line segment $[R_k, (0, i_k/2, n - i_k/2)]$).

Given a network N, the weight $u^{N\,f_0}_{z=k/2}$ is defined by: $u^{N\,f_0}_{z=k/2} = min\{w(e'), |e' \in E^{f_0}_{z=k/2} \cap N\}$. This weight represents the minimal necessary weight for passing from one side of the cut $E^{f_0}_{z=k/2}$ to the other part still remaining in the network N.

Consider the path p_2 (of the dual of the projected input graph) linking the face f_0 to the infinite face f_∞ and remaining in the network N_2. We have:

$$w(p_2) \geq \sum_{0 \leq k \leq 2t} w^{N_2}_{z=\frac{k}{2}} + \sum_{2t < k \leq k_0} u^{N_2\,f_0}_{z=\frac{k}{2}},$$

because the considered cut sets of edges are pairwise disjoints and p_2 contains at least an edge of each of these sets.

From Lemma 1, we have: $u^{N_2\,f_0}_{z=\frac{k}{2}} = w^{N_0}_{x=\frac{i_k}{2}}$, since an edge of minimal weight for the two corresponding edge sets ($E^{f_0}_{z=\frac{k}{2}} \cap N_2$ and $E_{x=\frac{i_k}{2}} \cap N_0$) is the edge of vector $(\frac{i_k}{2}, \frac{2t+1}{2}, \frac{2n-i_k-2t-1}{2})$. Moreover, $2t < k < 2k_0$ if and only if $2n - 2k_0 - j_1 \leq i_k \leq 2n - 2t - j_1$. From the definition of j_1 this exactly means $2n - 2k_0 - 2j_0 \leq i_k \leq 2n - 2t - 2j_0$, i.e. $2i_0 \leq i_k \leq 2n - 2t - 2j_0$. Therefore, $\sum_{2t<k\leq k_0} u^{N_2\,f_0}_{z=\frac{k}{2}} = \sum_{2i_0 \leq i \leq 2n-2t-2j_0} w^{N_0}_{x=\frac{i}{2}}$, which gives:

$$w(p_2) \geq \sum_{0 \leq k \leq 2t} w^{N_2}_{z=\frac{k}{2}} + \sum_{2i_0 \leq i \leq 2n-2t-2j_0} w^{N_0}_{x=\frac{i}{2}}$$

On the other hand, for $2t < j < 2j_0$, we have, from Lemma 11, $w^{N_1}_{y=\frac{j}{2}} = w^{N_0}_{x=\frac{2n-2t-j}{2}}$: an edge of minimal weight in the edge set $E_{y=\frac{j}{2}} \cap N_1$ is the edge e of vector $(\frac{2n-j-2t-1}{2}, \frac{j}{2}, \frac{2t+1}{2})$, an edge of minimal weight in the edge set $E_{x=\frac{2n-2t-j}{2}} \cap N_0$ is the edge e' of vector $(\frac{2n-j-2t}{2}, \frac{j+1}{2}, \frac{2t+1}{2})$, and $w(e) = w(e')$.

Thus, $\sum_{2t<j<2j_0} w^{N_1}_{y=\frac{j}{2}} = \sum_{2t<j<2j_0} w^{N_0}_{x=\frac{2n-2t-j}{2}} = \sum_{2n-2t-2j_0<i<2n-4t} w^{N_0}_{x=\frac{i}{2}}$, which gives:

$$w(p_1) \geq \sum_{0 \leq j \leq 2t} w^{N_1}_{y=\frac{j}{2}} + \sum_{2n-2t-2j_0<i<2n-4t} w^{N_0}_{x=\frac{i}{2}}.$$

We recall that

$$w(p_0) \geq \sum_{0 \leq i < 2i_0} w^{N_0}_{x=\frac{i}{2}}.$$

We have $w(E'_\phi) = w(p_0) + w(p_1) + w(p_2)$, thus, adding the three previous main inequalities, we get:

$$w(E'_\phi) \geq \sum_{0 \leq i < 2n-4t} w^{N_0}_{x=\frac{i}{2}} + \sum_{0 \leq j \leq 2t} w^{N_1}_{y=\frac{j}{2}} + \sum_{0 \leq k \leq 2t} w^{N_2}_{z=\frac{k}{2}}.$$

But $\sum_{0 \leq i < 2n-4t} w^{N_0}_{x=\frac{i}{2}}$ is exactly the weight of the path separating the 2-zone and the 1-zone for the function ψ, $\sum_{0 \leq j \leq 2t} w^{N_1}_{y=\frac{j}{2}}$ is exactly the weight of the path separating the 2-zone and the 0-zone for the function ψ, and $\sum_{0 \leq k \leq 2t} w^{N_2}_{z=\frac{k}{2}}$ is exactly the weight of the path separating the 1-zone and the 0-zone for the function ψ. Thus the second member of the equality is exactly $w(E'_\psi)$. We have: $w(E'_\phi) \geq w(E'_\psi)$. □

Lemma 4. *Let ϕ be a consensus function. There exists a consensus function ϕ' for which each the induced zones induced by ϕ' are connected, such that $inst(\phi') \leq inst(\phi)$.*

Proof. Actually, we prove a (little bit) stronger fact: if a zone induced by ϕ is not connected, then there exists a consensus function ϕ' such that $inst(\phi') < inst(\phi)$. If we apply this fact a sufficient number of times, then we will end up with consensus function with connected zones (obtaining the lemma), since the set of consensus functions is finite.

We call, for short, *0-domain* any connected component of the 0-zone (we define in a same way 1-domains and 2-domains). Assume that the 0-zone of ϕ is not connected and let D be a 0-domain which does not contain the vertex $(n, 0, 0)$. Let (i_1, j_1, k_1) be a vertex of D with j_1 minimal and (i_2, j_2, k_2) be a vertex of D with k_2 minimal.

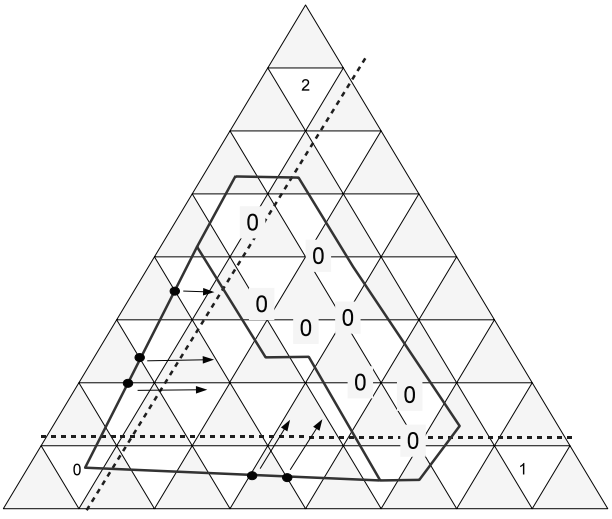


Fig. 4. The function ϕ' used in the tricky case of Lemma 4. The marked edges are some potential edges of $E_{\phi'} \setminus E_\phi$. Arrows indicate the way for fetching $i(e)$.

First assume that $j_1 \geq t+1$. If there exists an edge $e = \{v, v'\}$, bicolor for ϕ, with $v \in D$ and $\phi(v') = 2$, then the function ϕ_2, defined by $\phi_2(v) = 2$ for $v \in D$, and $\phi_2(v) = \phi(v)$ otherwise, is a consensus function such that $inst(\phi_2) < inst(\phi)$: we are done. Otherwise, each edge $e = \{v, v'\}$, bicolor for ϕ, with $v \in D$ is such that $\phi(v') = 1$. This enforces that, for any such edge, the 2-component of v is at least $t+1$. Thus the function ϕ_1, defined by $\phi_1(v) = 1$ for $v \in D$, and $\phi_2(v) = \phi(v)$ otherwise, is a consensus function, such that $inst(\phi_1) < inst(\phi)$: we are done.

A similar argument can be used if it is assumed that $k_2 \geq t+1$. It remains to treat the tricky case when $j_1 \leq t$ and $k_2 \leq t$. We state $V_{j_1,k_2} = \{(i,j,k) \in \mathbb{N}^3 \mid i+j+k = n, j_1 \leq j, k_2 \leq k\}$. In this case, since D is connected, one can fix a simple path p in D from (i_1, j_1, k_1) to (i_2, j_2, k_2). Let $support(p)$ denote the set of vertices appearing in p. From the Jordan curve theorem [9,13], the path p divides $V_{j_1,k_2} \setminus support(p)$ into two parts, both connected in the projected input graph. Let D_p be the part which contains the vertex $(n-2t, t, t)$ (this vertex is not element of $support(p)$ since it is not element of D). Notice that the 0-component of any vertex of D_p is at least $t+1$. Thus, the function ϕ', defined by $\phi'(v) = 0$ for $v \in D_p$, and $\phi'(v) = \phi(v)$ otherwise, is a consensus function (see Figure 4).

The set $E'_{\phi'} \setminus E'_\phi$ can be partitioned into two sets E_j and E_k, where $e = \{v, v'\}$ is element of V_k if v is a vertex of D_p whose 2-component is k_2 and v' is a vertex whose 2-component is $k_2 - 1$ such that $\phi(v') = 1$, and, on the symmetric way, $e = \{v, v'\}$ is element of V_j if v is a vertex of D_p whose 1-component is j_1 and v' is a vertex whose 1-component is $j_1 - 1$ such that $\phi(v') = 2$.

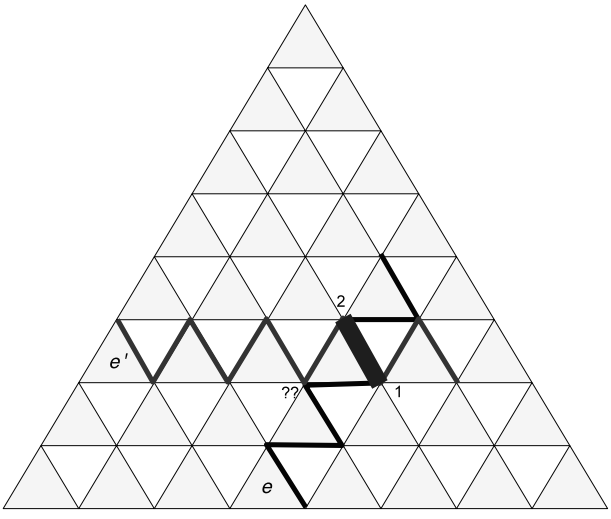


Fig. 5. Injectivity of the mapping i in Lemma 4, in the tricky case when i and j both are odd. If $i(e) = i(e')$, then $i(e)$ is the common edge. Each vertex v from e to the common edge is such that $\phi(v) = 1$ and, by symmetry, each vertex v' from e' to the common edge is such that $\phi(v') = 2$. Contradiction (at the vertex with question marks).

Take an edge $e = \{v, v'\}$ of V_k, and let $(\frac{2n-j-2k_2+1}{2}, \frac{j}{2}, \frac{2k_2-1}{2})$ be its vector. Let k' be the lowest integer such that $k' \geq 2k_2 - 1$ and the edge of vector $(\frac{2n-j-k'}{2}, \frac{j}{2}, \frac{k'}{2})$ is bicolor for ϕ. This last edge is denoted by $i(e)$ (informally, $i(e)$ is the edge of $E_{y=\frac{j}{2}}$ which allows to go out of the connected component containing e for ϕ). We have $j \geq 2t + 1$, since $i(e)$ is bicolor, thus we have $k' \leq 2n - 2t - 1$. Thus $2k_2 - 1 < k' < 2n - 2t - 1$, which gives, from Lemma 1, $w(e) < w(i(e))$. Moreover, $i(e)$ is not element of $E_{\phi'}$ since, by definition, both vertices of $i(e)$ are elements of $D_p \cup support(p)$.

In a similar way, for each edge $e = \{v, v'\}$ of V_j, let $(\frac{2n-2j_1-k+1}{2}, \frac{2j_1-1}{2}, \frac{k}{2})$ be its vector. Let j' be the lowest integer such that $j' \geq 2j_1 - 1$ and the edge of vector $(\frac{2n-j'-k}{2}, \frac{j'}{2}, \frac{k}{2})$ is bicolor for ϕ. This last edge is denoted by $i(e)$ (informally, $i(e)$ is the edge of $E_{z=\frac{k}{2}}$ which allows to go out of the connected component containing e for ϕ). We have: $w(e) < w(i(e))$ and moreover, $i(e)$ is not element of $E_{\phi'}$.

The other edges of $E_{\phi'}$ are contained in E_ϕ. For these edges, we state: $i(e) = e$. We claim that the mapping i is injective. To see it, we only have to check that if e is an edge of V_k and e' is an edge of V_j, then $i(e) \neq i(e')$. This can be easily done by a case by case analysis according to the parity of integers j and k such that $e' \in E_{y=\frac{j}{2}}$ and $e' \in E_{z=\frac{k}{2}}$ (see the argument in Figure 5).

In any case, we have: $w(e) \leq w(i(e))$. Moreover, we have: $V_k \neq \emptyset$ since, otherwise, we have $\phi(v) = 0$ for any v of the type (i, j, k_2), which contradicts the fact that the connected domain D does not contains $(n, 0, 0)$. Thus there is at least an edge e of $E'_{\phi'}$ for which $w(e) < w(i(e))$. Thus, by addition, we get $w(E'_{\phi'}) < w(E'_\phi)$, which is the result. □

References

1. Becker, F., Rajsbaum, S., Rapaport, I., Rémila, E.: Average Binary Long-Lived Consensus: Quantifying the Stabilizing Role Played by Memory. In: Shvartsman, A.A., Felber, P. (eds.) SIROCCO 2008. LNCS, vol. 5058, pp. 48–60. Springer, Heidelberg (2008)
2. Berman, P., Garay, J.: Cloture votes: $n/4$-resilient distributed consensus in $t + 1$ rounds. Math. Sys. Theory 26(1), 3–19 (1993)
3. Chandrakasan, A.P., Brodersen, R.W.: Low power digital CMOS design. Kluwer Academic Publishes, Dordrecht (1995)
4. Cohen, D.I.A.: On the Sperner lemma. J. Combin. Theory 2, 585–587 (1967)
5. Dalhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, P.: The Complexity of Multiterminal Cuts. SIAM J. on Computing 23(4), 864–894 (1994)
6. Davidovitch, L., Dolev, S., Rajsbaum, S.: Stability of Multi-Valued Continuous Consensus. SIAM J. on Computing 37(4), 1057–1076 (2007)
7. Dolev, S.: Self-Stabilization, March 2000. MIT Press, Cambridge (2000)
8. Dolev, S., Rajsbaum, S.: Stability of Long-lived Consensus. J. of Computer and System Sciences 67(1), 26–45 (2003); Preliminary version in Proc. of the 19th Annual ACM Symp. on Principles of Distributed Computing (PODC 2000), pp. 309–318 (2000)
9. Jordan, C.: Cours d'analyse, Ecole Polytechnique, pp. 587–594 (1887)
10. Kahn, J., Kalai, G., Linial, N.: The Influence of Variables on Boolean Functions. In: Proc. of the IEEE FOCS, pp. 68–80 (1988)
11. Kutten, S., Masuzawa, T.: Output Stability Versus Time Till Output. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 343–357. Springer, Heidelberg (2007)
12. Kopetz, H., Veríssimo, P.: Real Time and Dependability Concepts. In: Mullender, S. (ed.) Distributed Systems, ch. 16, pp. 411–446. ACM Press, New York (1993)
13. Maehara, R.: The Jordan Curve Theorem via the Brouwer Fixed Point Theorem. American Mathematical Monthly 91, 641–643 (1984)
14. Musoll, E., Lang, T., Cortadella, J.: Exploiting the locality of memory references to reduce the address bus energy. In: Proc. of the Int. Symp. on Low Power Electronics and Design, August 1997, pp. 202–207 (1997)
15. Norris, J.R.: Markov Chains, October 1998. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (1998)
16. Sperner, E.: Neuer Beweis f .ur die Invarianz der Dimensionszahl und des Gebietes, Abh. Math. Sem. Univ. Hamburg 6, 265–272 (1928)
17. Su, C.-L., Tsui, C.-Y., Despain, A.M.: Saving power in the control path of embedded processors. IEEE Design & Test of Comp., 24–30 (1994)

Algorithms for Extracting Timeliness Graphs*

Carole Delporte-Gallet[1], Stéphane Devismes[2],
Hugues Fauconnier[1], and Mikel Larrea[3]

[1] Université Paris Diderot
LIAFA
{Carole.Delporte,Hugues.Fauconnier}@liafa.jussieu.fr
[2] Université Joseph Fourier, Grenoble I
VERIMAG UMR 5104
Stephane.Devismes@imag.fr
[3] University of the Basque Country, UPV/EHU
Mikel.Larrea@ehu.es

Abstract. We consider asynchronous message-passing systems in which some links are timely and processes may crash. Each run defines a *timeliness graph* among correct processes: (p, q) is an edge of the timeliness graph if the link from p to q is timely (that is, there is a bound on communication delays from p to q). The main goal of this paper is to approximate this timeliness graph by graphs having some properties (such as being trees, rings, ...). Given a family S of graphs, for runs such that the timeliness graph contains at least one graph in S then using an *extraction algorithm*, each correct process has to converge to the same graph in S that is, in a precise sense, an approximation of the timeliness graph of the run. For example, if the timeliness graph contains a ring, then using an extraction algorithm, all correct processes eventually converge to the same ring and in this ring all nodes will be correct processes and all links will be timely.

We first present a general extraction algorithm and then a more specific extraction algorithm that is communication efficient (*i.e.*, eventually all the messages of the extraction algorithm use only links of the extracted graph).

1 Introduction

We consider partially synchronous models like in [6] or [7] in which some processes may crash. In such systems some links are timely, meaning that the communication delays are bounded [2], and some other are not. Generally, these timeliness properties of links have been used to solve the consensus problem as in [7] or to implement failure detectors like Ω that realizes an eventual election of a correct process (*e.g.*, [1,4,10,11,12]). In this paper we are more specifically interested in detecting the timeliness of the links in order to approximate the

* This work has been supported in part by the ANR project *SHAMAN*, the INRIA project GANG, and the Basque Government (grant MV-2009-1-10).

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 127–141, 2010.

timeliness relation on links in each run. If processes are able to eventually determine which links are timely, then avoiding to use non timely links could help to improve the efficiency of the communication that can be particularly interesting for routing algorithms.

More precisely, each run of the system eventually converges to a *timeliness graph* whose nodes are the correct processes and directed edges are the timely edges among correct processes, and an *extraction algorithm* is an algorithm such that all correct processes eventually agree on an identical graph that approximates the timeliness graph.

For example, assume the system ensures that there is at least one correct process that communicates in a timely way with all other processes, such a process is an *eventual source* [2] and it could be interesting for the processes to choose and agree on such an eventual source. This way, we not only realize an eventual leader election but also the chosen leader is able to communicate in a timely way with the rest of correct processes.

If we assume now that instead of an eventual source there is an eventual *root* in the system, that is a correct process that may communicate with every process by a communication path using only timely links, then choosing and agreeing on such an eventual root realizes an eventual leader election (the root is the eventual leader) but this also enables to ensure a routing of all messages from the root to any other processes using only timely links.

In the same way, if the system ensures that there is always a cycle containing all correct processes in the timeliness graph of the run, then choosing and agreeing on one such cycle enables to eventually build a ring between all correct processes that uses only timely links. Note that in this case the processes eventually agree on the list of all correct processes too, as a consequence we obtain a failure detector $\Diamond P$ [5].

More precisely, consider some structural property $\mathcal{P}$ of graphs (like being a star, a ring, a tree, a complete graph...). An algorithm *extracting* a graph G verifying $\mathcal{P}$ has to ensure that (1) all the correct processes eventually agree on G, (2) all the correct processes are nodes of G and (3) G is an "approximation" of the timeliness relation of the run. Actually, "approximation" means that the subgraph of G induced by the correct processes is obtained from a directed cut (dicut) [1] of G and is a subgraph of the timeliness graph.

Contributions. In this paper, we first introduce and specify the problem of extraction of graphs in some set $\mathcal{X}$. We consider only systems in which a solution may exist: in all runs there is at least one graph in $\mathcal{X}$ that is compatible with the run.

We prove that this problem cannot be solved for some set of graphs and we give a sufficient condition on the set of graphs to be extracted. This condition is rather simple: the set of graphs has to be closed by directed cut reduction. Then, we give an extraction algorithm for every set of graphs verifying this property.

[1] A directed cut (X, Y) of directed graph $G = \langle N, E \rangle$ is a partition of N such that there is no directed edge from Y to X.

Moreover, if the graphs in $\mathcal{X}$ are all strongly connected, the algorithm gives an exact extraction, that is, the set of nodes of the extracted graph is exactly the set of correct processes of the run. Reciprocally, we show that there exist sets of graphs that admit extraction but no exact extraction.

Besides, we show that finding an approximation is even so interesting: in the extracted graph any path between a pair of correct processes is only constituted of timely links. Hence, the approximation can be used to timely route messages, *e.g.*, in the previous example with a root, the approximation will give us a tree whose root is a correct process and with a path containing only correct processes from the root to every correct process.

One drawback of this algorithm is the fact that forever all correct processes have to send messages on all links. Hence if k is the number of correct processes $k(k-1)$ links will be used forever by the extraction algorithm. We are then interested in *communication efficient* [11] implementations of the extraction problem. That is, eventually all correct processes only send messages along the edges of the extracted graph. For example, consider the example of a system with a timeliness ring, eventually only $k-1$ links of the system are used. We propose an efficient extraction algorithm for sets of graphs containing at least one correct process with directed paths from this process to all correct processes.

Roadmap. In the next section, we define the model used in this paper and present some examples of systems. In Section 3, we define the extraction problem and give some of its properties. Our two algorithms are presented in Sections 4 and 5, respectively. Finally, we make some concluding remarks in Section 6.

Due to the lack of space, some technical proofs have been omitted. For further details, see the online technical report [8].

2 Informal Model

Graphs. We begin with some definitions and notations concerning graphs. For a directed graph $G = \langle N, E\rangle$, $Node(G)$ and $Edge(G)$ denote N and E, respectively. Given a graph G and a set $M \subseteq Node(G)$, $G[M]$ is the *subgraph* of G induced by M, *i.e.*, $G[M]$ is the graph $\langle M, Edge(G)[M]\rangle$ where $(p,q) \in Edge(G)[M]$ if and only if $p, q \in M$ and $(p,q) \in Edge(G)$.

The tuple (X,Y) is a *directed cut* (*dicut* for short) of G if and only if X and Y define a partition of $Node(G)$ and there is no directed edge $(y,x) \in Edge(G)$ such that $x \in X$ and $y \in Y$. We say that G' is a *dicut reduction* from G if there exists a dicut (X,Y) of G such that $G' = G[X]$. A set S of graphs is *dicut-closed* if and only if it is closed under dicut reduction, namely if $G \in S$ then all the graphs obtained by a dicut-reduction of G are in S.

Processes and Links. We consider distributed systems composed of n processes which communicate by message-passing through directed links. We denote the set of processes by $\Pi = \{p_1, ..., p_n\}$. We assume that the communication graph is complete, *i.e.*, for each pair of distinct processes (p,q), there is a directed link from p to q.

A process may fail by crashing, in which case it definitely stops its local algorithm. A process that never crashes is said to be *correct*, *faulty* otherwise.

The (directed) links are *reliable*, *i.e.*, every message sent through a link (p, q) is eventually received by q if q is correct and if a message m from p is received by q, m is received by q at most once, and only if p previously sent m to q.

The links being reliable, an implementation of the *reliable broadcast* [9] is possible. A reliable broadcast is defined with two primitives: `rbroadcast`$\langle m \rangle$ and `rdeliver`$\langle m \rangle$. Informally, after a correct process p invokes `rbroadcast`$\langle m \rangle$, all correct processes eventually `rdeliver`$\langle m \rangle$; after a faulty process p invokes `rbroadcast`$\langle m \rangle$, either all correct processes eventually `rdeliver`$\langle m \rangle$ or correct processes never `rdeliver`$\langle m \rangle$.

Timeliness. To simplify the presentation, we assume the existence of a discrete global clock. This is merely a fictional device: the processes do not have access to it. We take the range $\mathcal{T}$ of the clock's ticks to be the set of natural numbers.

We assume that every correct process p is *timely*, *i.e.*, there is a lower and an upper bound on the execution rate of p. Correct processes also have clocks that are not necessarily synchronized but we assume that they can accurately measure intervals of time.

A link (p, q) is *timely* if there is an unknown bound δ such that no message sent by p to q at time t may be received by q after time $t + \delta$.

A *timeliness graph* is simply a directed graph whose set of nodes are a subset of Π. The timeliness graph represents the timeliness properties of the links. Intuitively, for timeliness graph G, $Node(G)$ is the set of correct processes and (p, q) is in $Edge(G)$ if and only if the link (p, q) is timely.

Runs. An algorithm $\mathcal{A}$ consists of n deterministic (infinite) automata, one for each process; the automaton for process p is denoted $\mathcal{A}(p)$. The execution of an algorithm $\mathcal{A}$ proceeds as a sequence of process *steps*. Each process performs its steps atomically. During a step, a process may send and/or receive some messages and changes its state.

A run r of algorithm $\mathcal{A}$ is a tuple $r = \langle T, I, E, S \rangle$ where T is a timeliness graph, I is the initial state of the processes in Π, E is an infinite sequence of steps of $\mathcal{A}$, and S is a list of increasing time values indicating when each step in E occurred. A run must satisfy usual properties concerning sending and receiving messages. Moreover, we assume that (1) all correct processes make an infinite number of steps: $p \in Node(T)$ if and only if p makes an infinite number of steps in E and (2) the timeliness of links is deduced from the timeliness graph T: $(p, q) \in Edge(T)$ if and only if the link (p, q) is timely with respect to E and S.

In the following for run $r = \langle T, I, E, S \rangle$, $T(r)$ denotes T the timeliness graph of r, and $Correct(r)$ is the set of correct processes for the run r, namely, $Correct(r) = Node(T(r))$. Note that by definition, (p, q) is a timely link if and only if $(p, q) \in Edge(T)$.

Remark that in the definition given here a link may be timely even if no message is sent on the link. If link (p, q) is FIFO (*i.e.*, messages from p to q are received in the order they are sent) and p regularly sends messages to q, then

the timeliness of these messages implies the timeliness of the link itself. So in the following we always assume that links are FIFO.

2.1 Some Systems

We say that timeliness graph G is *compatible with timeliness graph* G' if and only if (1) $Node(G) = Node(G')$ and (2) $Edge(G) \subseteq Edge(G')$. By extension, timeliness graph G is *compatible with run* r if G is compatible with $T(r)$, the timeliness graph of r. Hence, timeliness graph G is compatible with run r if $Node(G)$ is the set of correct processes in r and if (p, q) is an edge of G then (p, q) is timely in r.

A *system* $\mathcal{X}$ is defined as a set of timeliness graphs. The set of runs of system $\mathcal{X}$ denoted $R(\mathcal{X})$ is the set of all runs r such that there exists a timeliness graph G in $\mathcal{X}$ compatible with r.

Below, we define the systems considered in this paper:

- $\mathcal{ASYNC}$ is the set of all timeliness graphs G such that $Edge(G) = \emptyset$. In $\mathcal{ASYNC}$ there is no timeliness assumption about links and $R(\mathcal{ASYNC})$ is the set of all runs in an asynchronous system.
- $\mathcal{COMPLETE}$ is the set of all complete graphs whose nodes are the subsets of Π.
- $\mathcal{STAR}$ is the set of all timeliness graphs with a *source*, *i.e.*, $G \in \mathcal{STAR}$ if and only if $Node(G) \subseteq \Pi$ and there exists $p_0 \in Node(G)$ (the center of the star or the source) such that $Edge(G) = \{(p_0, q) | q \in Node(G) \setminus \{p_0\}\}$. Clearly a run r is in $R(\mathcal{STAR})$ if and only if there is at least one *source* in r.
- $\mathcal{TREE}$ is the set of all timeliness graphs G that are rooted directed trees, *i.e.*, $|Edge(G)| = |Node(G)| - 1$ and there exists p_0 in $Node(G)$ such that $\forall q \in Node(G)$, there is a directed path of G from p_0 to q. Clearly a run r is in $R(\mathcal{TREE})$ if and only if there is at least one timely path from a correct process to all correct processes.
- $\mathcal{RING}$ is the set of all timeliness graphs G such that G is a directed cycle (a ring). Clearly a run r is in $R(\mathcal{RING})$ if and only if there is a timely (directed) cycle over all correct processes.
- $\mathcal{SC}$ is the set of all timeliness graphs that are strongly connected. Clearly, a run r is in $R(\mathcal{SC})$ if and only if there exists a (directed) timely path between each pair of distinct correct processes.
- $\mathcal{BIC}$ is the set of all timeliness graphs G such that for all p, $q \in Node(G)$, there exist at least two distinct paths from p to q. $\mathcal{BIC}$ corresponds to the set of 2-strongly-connected graphs. Clearly, a run r is in $R(\mathcal{BIC})$ if and only if there exists at least two distinct timely paths between each pair of distinct correct processes.
- $\mathcal{PAIR}$ is the set of all timeliness graphs G such that $Edge(G) = \{(p, q), (q, p)\}$ with $p, q \in Node(G)$ and $p \neq q$. Clearly, a run r is in $R(\mathcal{PAIR})$ if and only if there exists two distinct correct processes p and q such that (p, q) and (q, p) are timely links.

3 Extraction Algorithms

Given a system $\mathcal{X}$, the goal of an *extraction algorithm* is to ensure that in each run r in $R(\mathcal{X})$, all correct processes eventually agree on the same element of $\mathcal{X}$ and that this element is, in some precise sense, an approximation of the timeliness graph of run r.

For example, in $\mathcal{RING}$, all processes have to eventually agree on some ring and this ring has to be compatible with the timeliness graph of the run. In particular this ring contains all the correct processes. However, the compatibility relation may be too strong: In many systems, it is not possible to distinguish between a crashed process and a correct one, so the graph G on which the processes eventually agree may contain crashed processes and then the graph is not exactly compatible with the run. Then we weaken the compatibility and impose only that the subgraph of G induced by the set of correct processes of the run is a dicut reduction of the timeliness graph of the run.

We now formally define what an extraction algorithm is. First, in such an algorithm, every process p maintains a local variable G_p which contains a timeliness graph. Then, we say that an algorithm *extracts a timeliness graph in* $\mathcal{X}$ if and only if for every run r in $R(\mathcal{X})$ there is a timeliness graph G (called the *extracted graph*) such that:

- *Convergence:* for all correct processes p there is a time t after which $G_p = G$
- *Compatibility:* $G[Correct(r)]$ is compatible with $T(r)$
- *Closure:* $G[Correct(r)]$ is a dicut reduction of G or is equal to G
- *Validity:* G is in $\mathcal{X}$

Remark that for all systems that contain $\mathcal{ASYNC}$ there is a trivial extraction algorithm: for each run processes extract the graph G such that $Node(G) = \Pi$ and $Edge(G) = \emptyset$.

A more constrained version of the extraction problem is the following: an algorithm $\mathcal{A}$ *extracts exactly* timeliness graphs in $\mathcal{X}$ if for every run r in $R(\mathcal{X})$, the extracted graph G is compatible with $T(r)$. In this case, all correct processes eventually know the exact set of correct processes: it is the set of nodes of the extracted graph.

Some Results about Extraction Algorithms. First we show that an extraction algorithm may help to route messages using only timely links:

Lemma 1. *Let G be a graph extracted from run r, if (p, q) is in $Edge(G)$ and q is a correct process then p is correct.*

Proof. By contradiction, assume that p is not correct, then $(Correct(r), Node(G) - Correct(r))$ is not a dicut because $(p, q) \in Edge(G)$, $p \in Node(G) - Correct(r)$ and $q \in Correct(r)$, which contradicts the Closure property.

From this lemma and the Compatibility property, we deduce directly:

Proposition 1. *If $(p = q_0, \ldots, q_i, \ldots, q = q_m)$ is a path in the extracted graph and p and q are correct processes, then for every i such that $0 \leq i < m$ the link (q_i, q_{i+1}) is timely and process q_i is correct.*

From a practical point of view, this proposition shows that the extracted graph may be used to route messages between processes using only timely links: the route from p to q is a path in the extracted graph (if any). All intermediate nodes are correct processes and agree on the extracted graph and then on the path.

For example with $\mathcal{TREE}$, the tree extracted by the algorithm enables to route messages from the root of the tree to any other processes and the routing uses only timely links.

Generally, the main goal of the extraction algorithm is not only to extract a graph G in $\mathcal{X}$ but also to ensure that $G[Correct(r)]$ is in $\mathcal{X}$ (even if the processes do not know the set of correct processes). In particular, this property is ensured if $\mathcal{X}$ is dicut-closed: the Closure property implies that $G[Correct(r)]$ is in $\mathcal{X}$.

Among the systems we consider, only system $\mathcal{PAIR}$ is not dicut-closed: $H = \langle\{x\}, \emptyset\rangle$ is a dicut reduction of $G = \langle\{x, y, z\}, \{(y, z), (z, y)\}\rangle$ but is not in $\mathcal{PAIR}$. It is easy to verify that every other previously introduced system is dicut-closed. For these systems we obtain:

Proposition 2. *Consider any extraction algorithm for the system $\mathcal{X}$.*

- *If $\mathcal{X} = \mathcal{STAR}$, then the center of the extracted star is a correct process.*
- *If $\mathcal{X} = \mathcal{TREE}$, then the root of the extracted tree is a correct process.*
- *If $\mathcal{X} \in \{\mathcal{SC}, \mathcal{COMPLETE}, \mathcal{RING}, \mathcal{BIC}\}$, then the extraction is exact.*

Proof. For $\mathcal{STAR}$ and $\mathcal{TREE}$, all the dicut reductions of the extracted graph contain at least respectively the center and the root, then the restriction of the extracted graph contains at least these nodes, proving that they are correct processes.

There is no dicut for a strongly connected graph. Hence in $\mathcal{SC}$, there is no dicut reduction then by the Closure property the subgraph induced by the set of correct processes of the extracted graph is the extracted graph itself. $\mathcal{COMPLETE}$, $\mathcal{RING}$, and $\mathcal{BIC}$ are particular cases of systems only composed of strongly connected timeliness graphs.

An immediate consequence of Proposition 2 is that any extraction algorithm gives an implementation of eventual leader election (failure detector Ω) for systems $\mathcal{STAR}$ and $\mathcal{TREE}$ as well as an implementation of failure detector $\Diamond\mathcal{P}$ for systems $\mathcal{COMPLETE}$, $\mathcal{RING}$, $\mathcal{SC}$ and $\mathcal{BIC}$.

Due to the lack of space, the proofs of the two following propositions have been omitted. In the first proposition we show that extraction is not always possible. Actually, in the proof we exhibit some non dicut-closed systems, namely $\mathcal{PAIR}$, where no extraction algorithm can be implemented.

Proposition 3. *There exist some systems $\mathcal{X}$ for which there is no extraction algorithm.*

In the next section we show that for all dicut-closed systems there is an extraction algorithm. For systems like $\mathcal{STAR}$, $\mathcal{TREE}$ and $\mathcal{PAIR}$, there exists no *exact* extraction algorithm.

Proposition 4. *There exist some systems $\mathcal{X}$ for which there is an extraction algorithm and there is no exact extraction algorithm.*

4 An Extraction Algorithm

The aim of this section is to show that the dicut-closed property of a system is sufficient to solve the extraction problem. To that end, we propose in Figure 1 an extraction algorithm, called $\mathcal{A}(\mathcal{X})$, for dicut-closed systems $\mathcal{X}$.

The basic idea of Algorithm $\mathcal{A}(\mathcal{X})$ is to make processes select a graph that is compatible with the timeliness graph of the run. For this, each process maintains for each graph x in $\mathcal{X}$ an *accusation counter* $Acc[x]$. This counter infinitely grows if some correct process is not in x or if some directed edge of x is not timely. Then, $Acc[x]$ is bounded if and only if x contains all correct processes and all timely links between pairs of correct processes.

We implement accusation counters as follows. A process regularly blames all the graphs in $\mathcal{X}$ in which it is not a node: it increments the accusation counters of all these graphs. Note that if the process is correct this accusation is justified and if the process is not correct, after some time, the process being dead stops to increment the accusation counters. Moreover, each process regularly sends on its outgoing links *alive* messages. Each process p maintains an estimate of the communication delays for each incoming link ($\Delta[q]$ for the incoming link (q, p)). If it does not receive *alive* messages within these estimates on some incoming link it blames all timeliness graphs in $\mathcal{X}$ containing this link (*i.e.*, increments the accusation counters for these graphs). As the estimate of the communication delay may be too short, each time it is exceeded the process increases it for the link. In this way, if the link is timely, at some time the estimate will be greater than the bound on communication delay.

The accusation counters are broadcast by reliable broadcasts. Each time a process receives a new value of accusation counter it updates its own accusation counter to the maximum of the received values and its current values. Hence, if some timely graph stops to be blamed then all correct processes eventually agree on the value of its accusation counter.

By selecting the graph G with the lowest accusation value (to break ties, we assume a total order among the graphs of $\mathcal{X}$) if any, correct processes eventually agree on the same timeliness graph of $\mathcal{X}$, moreover we can prove that this graph contains (1) all the correct processes, and (2) all edges between correct processes are timely links. As a consequence, the Convergence, the Compatibility and the Validity properties of the extraction algorithm are ensured. Nevertheless, this graph can also contain faulty processes and edges between correct and faulty processes.

Consider now the Closure property. If G contains only correct processes then the Closure property is trivially satisfied. Otherwise, G contains $Correct(r)$ and a set F of faulty processes. In this case, $(Correct(r), F)$ is a dicut reduction of G: Indeed if there is an edge in G from a faulty process q to a correct process p, eventually the process p stops to receive messages from q and the accusation counter of G grows infinitively often. Hence, in all cases, the Closure property is satisfied.

Hence, if $\mathcal{X}$ is dicut-closed, Algorithm $\mathcal{A}(\mathcal{X})$ extracts a graph in $\mathcal{X}$. Moreover from Proposition 2, if all the graphs of $\mathcal{X}$ are strongly connected then the algorithm exactly extracts a graph in $\mathcal{X}$.

In the algorithm, each process p uses local timers, one per process. The timer of p dedicated to q is set (by setting `settimer`(q) to a positive value) to a time interval rather than absolute time. The timer is decremented until it expires. When the timer expires `timerexpire`(q) becomes *true*. Note that a timer can be restarted before it expires.

In the algorithm, we denote by $\prec$ the total order relation on $\mathcal{X}$ and by $\prec_{lex}$ (see Line 2) the total order relation defined as follows: $\forall x, y \in \mathcal{X}$, $\forall c_x, c_y \in \mathbb{N}$, $(c_x, x) \prec_{lex} (c_y, y) \equiv [c_x < c_y \vee (c_x = c_y \wedge x \prec y)]$.

```
Code for each process p
 1: Procedure updateExtractedGraph()
 2:     G ← x such that (Acc[x], x) = min_{≺lex} {(Acc[x'], x') such that x' ∈ 𝒳}

 3: On initialization:
 4: for all x ∈ 𝒳 do Acc[x] ← 0
 5: for all q ∈ Π \ {p} do
 6:     Δ[q] ← 1
 7:     settimer(q) ← Δ[q]
 8: updateExtractedGraph()
 9: start tasks 1 and 2

10: task 1:
11:     loop forever
12:         send⟨alive⟩ to every q ∈ Π \ {p} every K time
13:         rbroadcast⟨ACC,⊥,p⟩ every K time  /* to accuse graphs that do not contain p */

14: task 2:
15:     upon receive⟨alive⟩ from q do
16:         settimer(q) ← Δ[q]

17:     upon timerexpire(q) do
18:         rbroadcast⟨ACC, q, p⟩  /* to accuse graphs that contain the link (q, p) */
19:             Δ[q] ← Δ[q] + 1
20:             settimer(q) ← Δ[q]

21:     upon rdeliver⟨ACC,q,h⟩ do /* information from h */
22:         for all x ∈ 𝒳 do
23:             if q =⊥ then
24:                 if h ∉ Node(x) then Acc[x] ← Acc[x] + 1
25:             else
26:                 if (q, h) ∈ Edge(x) then Acc[x] ← Acc[x] + 1
27:         updateExtractedGraph()
```

Fig. 1. Algorithm $\mathcal{A}(\mathcal{X})$ extracts a graph in $\mathcal{X}$

A sketch of the correctness proof of $\mathcal{A}(\mathcal{X})$ is given below. In this sketch, we consider a run r of $\mathcal{A}(\mathcal{X})$ in dicut-closed system $\mathcal{X}$. We will denote by var_p^t the value of var of process p at time t.

We first notice that all variables $Acc_p[x]$ are monotonically increasing:

Lemma 2. *For all times t and t' such that $t \geq t'$, for all processes p, for all graphs x in $\mathcal{X}$, $Acc_p^t[x] \geq Acc_p^{t'}[x]$.*

Let $\sup(Acc_p[x])$ be the supremum of $Acc_p^t[x]$ for all t, we say that $Acc_p[x]$ is unbounded if $\sup(Acc_p[x])$ is equal to ∞ and bounded otherwise. As $Acc_p[x]$ is

also updated by reliable broadcast each time some process q modifies $Acc_q[x]$ we have:

Lemma 3. *For all correct processes p and q, for all graphs x in $\mathcal{X}$,* $\sup(Acc_p[x]) = \sup(Acc_q[x])$

Let $\sup(Acc[x])$ be the supremum $\sup(Acc_p[x])$ over all correct processes p of $Acc_p[x]$ (by Lemma 3, $\sup(Acc[x])$ is well-defined). If there is a least one $x \in \mathcal{X}$ such that $\sup(Acc[x])$ is bounded, then $\min\{\sup(Acc[x'])|x' \in \mathcal{X}\}$ is finite, hence G the graph such that $(Acc[G], G) = \min_{\prec_{lex}}\{(Acc[x'], x')|x' \in \mathcal{X}\}$ is well defined. Then all correct processes converge to the same graph:

Lemma 4. *If there exists x in $\mathcal{X}$ such that* $\sup(Acc[x])$ *is bounded then there is a time after which for every correct process p, G_p is G.*

Now we prove the Compatibility property. Consider any timeliness graph $x \in \mathcal{X}$ compatible with $T(r)$. Then there is a time t after which all faulty processes are dead and the estimates of communication delays are greater than the bounds of communication delays of timely links of the run. After time t, (1) as x contains all correct processes, no process will blame x because it is not a node of x, and (2) as all edges of x are timely, no process will blame x for one of its edges then:

Lemma 5. *If x in $\mathcal{X}$ is compatible with $T(r)$, then* $\sup(Acc[x])$ *is bounded.*

Reciprocally, let x be a timeliness graph of $\mathcal{X}$ that is not compatible with the run. If process p is correct and p is not in x, it regularly blames x then $\sup(Acc[x]) = \infty$. If process p is not correct there is a time t after which it does not send any *alive* message, and there is a time after the timers on p expire forever for all correct processes, then if p is in x, $Acc_p[x]$ is incremented infinitely often and $\sup(Acc[x]) = \infty$. In the same way if q is correct and (p, q) is not timely, by the fifo property of the link, the timer for p expires infinitely often for process q and if (p, q) is an edge of x then $Acc_q[x]$ is incremented infinitely often and $\sup(Acc[x]) = \infty$.

Then:

Lemma 6. *For every x in $\mathcal{X}$, if* $\sup(Acc[x])$ *is bounded then $x[Correct(r)]$ is compatible with $T(r)$.*

Lemma 4 and Lemma 5 prove the Convergence property. Let G be the timeliness graph such that for every correct process p eventually $G_p = G$. Hence by Lemma 6:

Lemma 7. *$G[Correct(r)]$ is compatible with $T(r)$.*

It remains to prove that G satisfies the Closure property: $G[Correct(r)]$ is a dicut reduction of G or is equal to G. As $G[Correct(r)]$ is compatible with $T(r)$, we have:

Lemma 8. *$Correct(r) \subseteq Node(G)$.*

Let $F = Node(G) - Correct(r)$. If F is empty the Closure property is trivially ensured. Consider now the case where F is not empty. F contains only faulty processes and $(Correct(r), F)$ is a partition of $G(Node)$. If there is an edge in $Edge(G)$ from a faulty process q to a correct process p, eventually the process p never receives a message from q and the accusation counter of G will be unbounded, contradicting the choice of G. So, we have:

Lemma 9. *If $F \neq \emptyset$ then $Edge(G) \cap (F \times Correct(r)) = \emptyset$.*

Hence, $(Correct(r), F)$ is a dicut of G.

Lemma 4 and Lemma 5 prove the Convergence property, Lemma 7 proves the Compatibility property and Lemma 9 proves the Closure property. Moreover, G is clearly in $\mathcal{X}$ proving the Validity. Proposition 2 shows that the extraction is exact when all graphs of $\mathcal{X}$ are strongly connected. Hence, we can conclude with the following theorem:

Theorem 1. *Let $\mathcal{X}$ be a dicut-closed system. Algorithm $\mathcal{A}(\mathcal{X})$ extracts a graph in $\mathcal{X}$. Moreover if all graphs of $\mathcal{X}$ are strongly connected, Algorithm $\mathcal{A}(\mathcal{X})$ exactly extracts a graph in $\mathcal{X}$.*

5 An Efficient Extraction Algorithm

In this section, we propose another extraction algorithm called $\mathcal{AF}(\mathcal{X})$ (Figures 2 and 3). This algorithm is efficient meaning that the (correct) processes eventually only send messages along the edges of the extracted graph.

$\mathcal{AF}(\mathcal{X})$ (exactly) extracts a timeliness graph from system $\mathcal{X}$, where (1) $\mathcal{X}$ is dicut-closed and (2) for all graphs $g \in \mathcal{X}$ there is some process p, called *root*, such that there is a directed path from p to every node of g. For example, $\mathcal{TREE}$ and $\mathcal{RING}$ systems have this property.

In the following, we refer to these systems as *dicut-closed systems with a root*. For every graph g in $\mathcal{X}$, the function $root(g)$ returns a root of g.

In the algorithm, every process p stores several values concerning the graphs $x \in \mathcal{X}$ such that $root(x) = p$: (1) $Acc[x]$ is the accusation counter of x whose goal is the same as in Algorithm 1, (2) $Prop[x]$ is a *proposition counter* whose goal will be explained later, and (3) $\Delta[x]$ gives the expected time for a message to go from p (the root of the x) to all the nodes of x.

Every process also maintains a set variable $Candidates$. Each element of this set is a 4-tuple composed of a graph x of $\mathcal{X}$ and the newest values of $Acc[x]$, $Prop[x]$, and $\Delta[x]$ known by the process (the exact values are maintained at $root(x)$). Each element in this set is called *candidate* and each process selects its extracted graph among the graphs in the candidate elements.

As in Algorithm 1:

(1) Each process p sends *alive* messages on its outgoing links and monitors its incoming links. However, we restrain here the *alive* message sendings: process p sends *alive* messages on its outgoing link (p, q) only if (p, q) is in a graph candidate.

(2) A graph candidate is blamed if (a) a correct process is not in the graph or (b) a process receives an out of date message through one of its incoming links. In both cases the candidate is definitely removed from the $Candidates$ sets of all processes. To achieve this goal the process sends an accusation message (ACC) using a reliable broadcast and uses an array $Heard$ that ensures that an identical candidate (that is, the same graph with the same accusation and proposition values) can never be added again. Moreover, upon delivery of an accusation message for graph x, $root[x]$ increments $Acc[x]$.

We now present different mechanisms used to obtain the efficiency.

For all graphs $x \in \mathcal{X}$, only the process $root(x)$ is allowed to propose x as a candidate to the rest. Each process p stores its better candidate in its variable me, that is, the least blamed graph x such that $root(x) = p$.

- If a process finds in $Candidates$ a better candidate than me, it removes me from $Candidates$.
- If a process finds that me is better, it adds me to $Candidates$ and sends a new message containing me (1) to all processes that are not in $Node(me)$, and (2) to immediate successors of p in me. The immediate successors in me add me to their $Candidates$ set and relay the new message, and so on. By the reliability of the links, every correct process that is not in me eventually receives this message and blames me.

These mechanisms are achieved by the procedure $updateExtractedGraph()$. This procedure is called each time a graph candidate is blamed or a new candidate is proposed. Note that the $Candidates$ set is maintained with the set $OtherCand$ (the candidates of other processes), a boolean $Local$ that is true when the process has a candidate, and me, the graph candidate.

A process p may give up a candidate without this candidate being blamed: in this case, p is the root of the candidate, it finds a better candidate in $OtherCand$,

```
Code for each process p
 1: Procedure updateExtractedGraph()
 2:     Let (a_min, min) = min_{≺lex}{(acc, c) such that (c, acc, −, −) ∈ OtherCand} ∪ {(∞, ∞)}
 3:     if (a_min, min) < (Acc[me], me) ∧ Local then  /* Give up me */
 4:         rbroadcast⟨ACC,me,Acc[me],Prop[me],Δ[me]⟩
 5:         Prop[me] ← Prop[me] + 1
 6:         Local ← false
 7:     Candidates ← OtherCand
 8:     me ← x such that (a, x) = min_{≺lex}{(acc, c) such that c ∈ X ∧ root(c) = p}
 9:     if (Acc[me], me) < (a_min, min) ∧ Local = false then  /* Propose me */
10:         Local ← true
11:         Candidates ← Candidates ∪ {(me, Acc[me], Prop[me], Δ[me])}
12:         send⟨new,me,Acc[me],Prop[me],Δ[me]⟩ to every process not in Node(me)
13:         for all h ∈ Π \ {p} do
14:             if (h,p)∈ Edge(me) then
15:                 Δ[h]← max(Δ[h], Δ[me])
16:                 settimer(h) ← Δ[h]
17:             if (p,h)∈ Edge(me) and h ≠ root(me) then
18:                 send⟨new,me, Acc[me], Prop[me], Δ[me]⟩ to h
19:     G ← x such that (a, x) min_{≺lex}{(a′, x′) such that (x′, a′, p′, d′) ∈ Candidates}
```

Fig. 2. Procedure updateExtractedGraph of Algorithm $\mathcal{AF}(\mathcal{X})$

```
Code for each process p
20: On initialization:
21: for all x ∈ 𝒳 such that root(x) = p do
22:       Acc[x] ← 0; Prop[x] ← 0; Δ[x] ← n
23: for all x ∈ 𝒳 such that root(x) ≠ p do Heard[x] ← (−1, −1)
24: for all q ∈ Π \ {p} do Δ[q] ← 1
25: OtherCand ← ∅
26: Local ← false
27: me ← min{x such that x ∈ 𝒳 ∧ root(x) = p}
28: updateExtractedGraph()
29: start tasks 1 and 2

30: task 1:
31:       loop forever
32:             send⟨alive⟩ to every process q such that ∃(x,-,-,-)∈ Candidates and (p, q) ∈ Edge(x)
    every K time

33: task 2:
34:       upon receive⟨alive⟩ from q do
35:             settimer(q) ← Δ[q]

36:       upon timerexpire(q) do  /* Link (q, p) is not timely, blame all candidates that contain
    (q, p) */
37:             for all (x, a, pr, d) ∈ OtherCand such that (q, p) ∈ Edge(x) do
38:                   rbroadcast⟨ACC,x,a,pr,d⟩
39:             if (q, p) ∈ Edge(me) then
40:                   rbroadcast⟨ACC,me,Acc[me],Prop[me],Δ[me]⟩

41:       upon receive⟨new, x, a, pr, d⟩ from q do /* Proposition of a new candidate */
42:             if p ∉ Node(x) then  /* Blame x that does not contain p */
43:                   rbroadcast⟨ACC,x,a,pr⟩
44:             else
45:                   newCand ← false
46:                   if (x, −, −, −) ∉ OtherCand and Heard(x) < (a, pr) then  /* New candidate */
47:                         newCand ← true
48:                   if ∃(x, a_c, pr_c, d_c) ∈ OtherCand with (a_c, pr_c) < (a, pr) then  /* New candidate
    */
49:                         OtherCand ← OtherCand \ (c, a_c, pr_c, d_c)
50:                         newCand ← true
51:                   if newCand then
52:                         OtherCand ← OtherCand ∪ (x, a, pr, d)
53:                         updateExtractedGraph()
54:                         Heard[x] ← (a, pr)
55:                         for all h ∈ Π \ {p} do
56:                               if (h,p)∈ Edge(x) then
57:                                     Δ[h]← max(Δ[h], d)
58:                                     settimer(h)← Δ[h]
59:                               if (p,h)∈ Edge(x) and h ≠ root(x) then send⟨new, x, a, pr, d⟩ to h

60:       upon rdeliver⟨ACC,x,a,pr,d⟩ do
61:             if root(x) = p then
62:                   if x = me ∧ a = Acc[me] ∧ pr = Prop[me] then  /* Check if the accusation is up
    to date */
63:                         Acc[me] ← Acc[me] + 1; Δ[me] ← Δ[me] + 1
64:                         Local ← false
65:             else
66:                   OtherCand ← OtherCand \ (x, a, pr, d)
67:                   if Heard[x] < (a, pr) then Heard[x] ← (a, pr)
68:             updateExtractedGraph()
```

Fig. 3. Algorithm $\mathcal{AF}(\mathcal{X})$ that efficiently extracts a graph in $\mathcal{X}$

and removes me from $Candidates$. Then, p must not increment $Acc[me]$ when it receives accusations caused by this removing, indeed these accusations are not due to delayed messages. That is the goal of the proposition counter ($Prop$): in $Prop[x]$, $root(x)$ counts the number of times it proposes x as candidate and includes this value in each of its new messages (to inform other processes of the current value of the counter). Hence, when q wants to blame x, it now includes its own view of $Prop[x]$ in the accusation message. This accusation will be considered as legitimate by $root[x]$ (that is, will cause an increment of $Acc[x]$) only when the proposition counter inside the message matches $Prop[x]$. Also, whenever $root[x]$ removes x from $Candidates$, $root[x]$ increments $Prop[x]$ and does not send the new value to the other processes. In this way accusations due to this removing will be ignored.

For any timely candidate, the accusation counter will be bounded and its proposition counter increased each time it is proposed. In this way the graph with the smallest accusation and proposition values eventually remains forever in the $Candidates$ set of all correct processes and it is chosen as extracted graph. (This is done in the procedure $updateExtractedGraph()$.) Moreover, eventually all other candidates are given up and it remains only this graph in $Candidates$. In this way, only *alive* messages are sent and they are sent along the directed edges of the extracted graph ensuring the efficiency.

The following theorem states the correctness of $\mathcal{AF}(\mathcal{X})$. For space consideration, its proof has been omitted.

Theorem 2. *Let $\mathcal{X}$ be a dicut-closed system with a root. Algorithm $\mathcal{AF}(\mathcal{X})$ efficiently extracts a graph in $\mathcal{X}$. Moreover if all graphs of $\mathcal{X}$ are strongly connected, Algorithm $\mathcal{AF}(\mathcal{X})$ efficiently and exactly extracts a graph in $\mathcal{X}$.*

6 Conclusion

Failure detector implementations in partially synchronous models generally use the timeliness properties of the system to approximate the set of correct (or faulty) processes. In some way, the extraction problem is a kind of generalization: instead of only searching the set of correct processes, here we try to extract also information about the timeliness of links. Besides, our solutions are based on already existing mechanisms used in failure detectors implementations as in [2,3].

Information about the timeliness of links is useful for efficiency of fault-tolerant algorithms. In particular, in any extracted graph, any path between a pair of correct processes is only constituted of timely links. This property is particulary interesting to get efficient routing algorithms.

We gave an extraction algorithm for dicut-closed sets of timeliness graphs. Moreover, we proved that the extraction is exact when all the timeliness graphs are also strongly connected.

Given dicut-closed timeliness graphs that contain a root, we shown how to efficiently extract a graph from it. By efficiency we mean giving a solution where eventually messages are only sent over the links of the extracted graph.

It is important to note that the main purpose of the algorithms we proposed is to show the feasability of the extraction under some conditions. So, the complexity of our algorithms was not the main focus of this paper.

As a consequence, our algorithms are somehow unrealistic because of their high complexity. Giving more practical solutions will be the purpose of our future works.

Acknowledgments. We are grateful to members of the *Graph* team of the *LIAFA* Lab for the helpful discussions and their interesting suggestions.

References

1. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Stable leader election. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 108–122. Springer, Heidelberg (2001)
2. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing omega with weak reliability and synchrony assumptions. In: PODC, pp. 306–314 (2003)
3. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Communication-efficient leader election and consensus with limited link synchrony. In: Chaudhuri, S., Kutten, S. (eds.) PODC, pp. 328–337. ACM, New York (2004)
4. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing omega in systems with weak reliability and synchrony assumptions. Distributed Computing 21(4), 285–314 (2008)
5. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43(2), 225–267 (1996)
6. Dolev, D., Dwork, C., Stockmeyer, L.J.: On the minimal synchronism needed for distributed consensus. Journal of the ACM 34(1), 77–97 (1987)
7. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. Journal of the ACM 35(2), 288–323 (1988)
8. Delporte Gallet, C., Devismes, S., Fauconnier, H., Larrea, M.: Algorithms For Extracting Timeliness Graphs, http://hal.archives-ouvertes.fr/hal-00454388/en/
9. Hadzilacos, V., Toueg, S.: A modular approach to fault-tolerant broadcasts and related problems. Tech. Rep. TR 94-1425, Department of Computer Science, Cornell University (1994)
10. Hutle, M., Malkhi, D., Schmid, U., Zhou, L.: Chasing the weakest system model for implementing omega and consensus. IEEE Trans. Dependable Sec. Comput. 6(4), 269–281 (2009)
11. Larrea, M., Arévalo, S., Fernández, A.: Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In: Jayanti, P. (ed.) DISC 1999. LNCS, vol. 1693, pp. 34–48. Springer, Heidelberg (1999)
12. Mostéfaoui, A., Mourgaya, E., Raynal, M.: Asynchronous implementation of failure detectors. In: DSN, pp. 351–360. IEEE Computer Society, Los Alamitos (2003)

Distributed Tree Comparison with Nodes of Limited Memory*

Emanuele Guido Fusco[1] and Andrzej Pelc[2,**]

[1] Computer Science Department
Sapienza, University of Rome, via Salaria 113 – 00198 Rome, Italy
fusco@di.uniroma1.it
[2] Département d'informatique
Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada
pelc@uqo.ca

Abstract. We consider the task of comparing two rooted trees with port labelings. Roots of the trees are joined by an edge and the comparison has to be carried out distributedly, by exchanging messages among nodes. If the two trees are isomorphic, all nodes must finish in a state *YES*, otherwise they have to finish in a state *NO* and break symmetry, nodes of one tree getting label 0 and of the other – label 1. Nodes are modeled as identical automata, and our goal is to establish trade-offs between the memory size of such an automaton and the efficiency of distributed tree comparison, measured either by the time or by the number of messages used for communication between nodes. We consider both the synchronous and the asynchronous communication and establish exact trade-offs in both scenarios. For the synchronous scenario we are concerned with memory vs. time trade-offs. We show that if the automaton has x bits of memory, where $x \geq c \log n$, for a small constant c, then the optimal time to accomplish the comparison task in the class of trees of size at most n and of height at most $h > 1$ is $\Theta(\max(h, n/x))$. For the asynchronous scenario we study memory vs. number of messages trade-offs. We show that if the automaton has x bits of memory, where $n \geq x \geq c \log \Delta$, then the optimal number of messages to accomplish the comparison task in the class of trees of size at most n and of maximum degree at most Δ is $\Theta(n^2/x)$.

1 Introduction

We consider the task of comparing two rooted trees with port labelings. There are two disjoint rooted trees $T = (V, E)$ and $T' = (V', E')$ whose roots r and r' are joined by an edge permitting communication between these trees. Ports at each node v are labeled $0, \ldots, d(v) - 1$, where $d(v)$ is the degree of node v, and

* This work was done during the visit of Emanuele G. Fusco at the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

** Partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 142–156, 2010.

these labelings are arbitrary, with port numbers $d(r)$ and $d(r')$ corresponding to the joining edge at roots r and r', respectively. Trees (T, r) and (T', r') are *isomorphic*, if there is a bijection $f : V \longrightarrow V'$, such that $f(r) = r'$, u is adjacent to v, if and only if $f(u)$ is adjacent to $f(v)$, and the port number corresponding to edge $\{u, v\}$ at node u is equal to the port number corresponding to edge $\{f(u), f(v)\}$ at node $f(u)$. The aim of the tree comparison task is the following: if the two trees are isomorphic, all nodes must finish in a state *YES*, otherwise they have to finish in a state *NO* and break symmetry: nodes of one tree get label 0 and of the other – label 1.

Nodes of the two rooted trees to be compared are modeled as identical input/output automata that communicate by sending and receiving messages. Our goal is to establish trade-offs between the memory size of such an automaton and the efficiency of distributed tree comparison measured either by the time or by the number of messages used for communication between nodes. We consider both the synchronous and the asynchronous communication and establish exact trade-offs in both scenarios.

One of the most important applications of the tree comparison task is leader election in trees. Consider any tree without labels of nodes, but with port labelings. A tree has either a central node or a central edge. Which of these cases occurs can be easily checked in a distributed way. Starting from leaves, the tree can be pruned by first removing all leaves, then removing all leaves in the resulting tree, and so on, until a unique node or two adjacent nodes remain. In the first case this unique node is the central node, and in the second case the two adjacent nodes are joined by the central edge. In the first case, the central node becomes the leader. In the second case, leader election is possible, if and only if the subtrees rooted at both endpoints of the central edge are not isomorphic. If they are not isomporhic, the result of tree comparison solves leader election: the endpoint that got label 1 is the leader. Hence leader election in trees can be reduced to tree comparison.

Our results. For the synchronous scenario we are concerned with memory vs. time trade-offs. We show that if the automaton has x bits of memory, where $x \geq c \log n$, for a small constant c, then the optimal time to accomplish the comparison task in the class of trees of size at most n and of height at most $h > 1$ is $\Theta(\max(h, n/x))$. For the asynchronous scenario we study memory vs. number of messages trade-offs. We show that if the automaton has x bits of memory, where $n \geq x \geq c \log \Delta$, then the optimal number of messages to accomplish the comparison task in the class of trees of size at most n and of maximum degree at most Δ is $\Theta(n^2/x)$.

Related work. Tree comparison is a symmetry-breaking task, closely related to leader election. Leader election was first studied for the ring, under the assumption that all labels are distinct. A synchronous algorithm, based on comparisons of labels, and using $O(n \log n)$ messages was given in [8]. It was proved in [7] that this complexity is optimal for comparison-based algorithms. On the other hand, the authors showed an algorithm using a linear number of messages but

requiring very large running time. An asynchronous algorithm using $O(n \log n)$ messages was given, e.g., in [15] and the optimality of this message complexity was shown in [4]. Leader election in radio networks has been studied, e.g., in [9,10,14] and randomized leader election in [17].

Many authors [1,2,3,6,11,12,16,18,20] studied various computing problems in anonymous networks, whose nodes do not have labels, similarly as in our scenario. In particular, [5,20] characterize networks in which leader election can be achieved when nodes are anonymous. In [20] other important computing problems, such as the spanning tree construction and topology recognition, are studied in such networks. In [19] the authors study the problem of leader election in general networks, under the assumption that labels are not unique. They characterize networks in which this can be done and give an algorithm which performs election when it is feasible. They do not attempt to minimize the number of messages.

Tree canonization is a task related to tree comparison: the input is a rooted tree without port labelings and such a tree should get a unique isomorphism invariant name. The memory size needed for centralized execution of this task has been investigated in [13]. The author shows a centralized algorithm working in logarithmic space that decides if two rooted trees without port labelings are isomorphic.

To the best of our knowledge, the present paper is the first to study memory vs. time and memory vs. communication complexity trade-offs in symmetry breaking tasks.

The model. Each node is a copy of the same input/output automaton $\mathcal{A}$ which is a quadruple $(\mathcal{S}, Q, \pi, \lambda)$, where $\mathcal{S}$ is a finite set of states, Q is the input/output alphabet, $\pi : \mathcal{S} \times Q \longrightarrow \mathcal{S}$ is the state transition function, and $\lambda : \mathcal{S} \longrightarrow Q$ is the output function. The alphabet Q is the family of finite sets of couples $\{(i_1, m_{i_1}), \ldots, (i_k, m_{i_k})\}$, where i_j are non-negative integers and m_{i_j} are finite binary strings called *messages*. All nodes start in the same state S_0, called the *initial state*. Consider a node v that is in a state S. Let $\lambda(S) = \{(i_1, m_{i_1}), \ldots, (i_k, m_{i_k})\}$ be the output corresponding to state S. Node v sends message m_{i_j} on port i_j, for all its ports. If there is no pair (i_j, m_{i_j}) in $\lambda(S)$, no message is sent on port i_j, and all messages m_i, where i is not a port number, are ignored.

At any time, a node v that is currently in state S, can get a set of messages $m_{j_1}, \ldots, m_{j_s}$, on ports $j_1, \ldots, j_s$, respectively. The set $q = \{(j_1, m_{j_1}), \ldots, (j_s, m_{j_s})\}$ becomes an input symbol and the node transits to state $S' = \pi(S, q)$.

We consider both the synchronous and the asynchronous scenario. In the first one, time is slotted and all actions are carried out in *rounds*: in a given round a node in state S sends the appropriate messages that arrive to the corresponding neighbors in the same round, and every node after receiving messages in a round transits to the new state in the next round. In this scenario, the *time* of carrying out a comparison task for a given instance is the number of rounds it takes for this instance. In the asynchronous scenario, all actions can take arbitrarily long finite time, scheduled by an adversary. In particular, messages sent by nodes can arrive at different times to different neighbors. We only assume that a node in a

state S sends all messages prescribed by $\lambda(S)$ before transiting to a new state, and that whenever a node is in state S at time t and gets some messages at time $t' \geq t$, forming an input symbol q, then the node transits to state $\pi(S, q)$ before reacting to any messages received at some time $t'' > t'$.

There are three pairwise disjoint sets of states included in $\mathcal{S}$: the sets YES, NO_0 and NO_1. If the compared trees are isomorphic, each node of both trees should eventually enter a state from the set YES. Otherwise, each node of one of the trees must enter a state from the set NO_0 and each node of the other tree must enter a state from the set NO_1. Once a node enters a state in one of these sets, it remains forever in the set (although it may change states). More formally, for any $q \in Q$ and any states $S' \in YES$, $S'' \in NO_0$ and $S''' \in NO_1$, we have $\pi(S', q) \in YES$, $\pi(S'', q) \in NO_0$ and $\pi(S''', q) \in NO_1$.

We say that an automaton $\mathcal{A}$ solves the comparison problem in the class $\mathcal{C}$ of trees, if this task can be carried out for every pair of trees from $\mathcal{C}$ in which copies of $\mathcal{A}$ are placed in every node.

The memory of an automaton is measured by the number of states, or equivalently by the number of bits on which these states can be encoded. An automaton with K states requires $\Theta(\log K)$ bits of memory.

Due to lack of space, proofs of several results are omitted.

2 Preliminaries

Consider the set $\mathcal{T}_0$ of rooted trees where each node has label 0 at the port leading to its parent. Such a n-node tree can be encoded by a binary string of length $2n - 2$. This is done by performing a depth first visit of the tree, driven by increasing order of port numbers at each node, and writing a 1 every time an edge is traversed going down, and a 0 every time an edge is traversed going up. A tree $T \in \mathcal{T}_0$ can be reconstructed from its code as follows. Start from the root, making it the current node. In every step of the reconstruction we have a suffix σ of the code and a current node v. In the first step σ is the code. If the first element of σ is a 1, attach a new child to v, label the port connecting v to this child with the smallest port number not yet assigned at node v. Also assign label 0 to the port connecting the child to v. The child becomes the current node at the next step. If the first element of σ is a 0, the parent of v becomes the current node at the next step. In both cases, the first element of σ is removed.

A string s of length $2n - 2$ belongs to the set $\mathcal{C}_n$ of *well formed codes*, if and only if it has $n - 1$ ones, $n - 1$ zeroes, and no prefix of s contains more zeroes than ones. The coding and decoding functions described above define a bijection between the set $\mathcal{C}_n$ and the set of all n-node trees in $\mathcal{T}_0$. This is a subset of the trees we want to handle, as in general the port number $p(v)$ leading to the parent of a node v is arbitrarily chosen between 0 and $d(v) - 1$. Hence, we augment the code by inserting the port numbers leading to the parent of each internal node in the following way. The port number $p(v)$ of an internal node v is inserted, surrounded by | symbols, after the digit 1 corresponding to the first visit, in the encoding process, of node v. Denote such an augmented code of a tree T by

$B(T)$. From now on it is called the code of T. As symbol | has been added to the code, resulting in a ternary alphabet, we use 2 bits to represent each symbol in code $B(T)$ without ambiguities.

Proposition 1. *The length of the code $B(T)$ of a n-node tree T is $O(n)$.*

3 Trade-Offs between Memory and Time

In this section we consider the synchronous scenario and establish trade-offs between the memory size of an automaton and the time needed to accomplish the tree comparison task. Consider the class of trees with at most n nodes and of height at most h. Suppose that the automaton $\mathcal{A}$ placed at each node of the compared trees has x bits of memory, where $x \geq c \log n$, for a small constant c, whose value will be specified later. First notice that if $h = 1$, then trees in the considered class are stars. Two stars can be compared in constant time, regardless of the memory of the automaton, provided that it is at least logarithmic in n. Hence from now on we assume that $h > 1$.

Let $B(T)$ and $B(T')$ be the codes of trees T and T'. Our algorithm for comparing trees will make use of these codes. While comparing tree T and tree T', codes of subtrees of T and T' are built, in a bottom-up fashion, starting from the leaves. The comparison between $B(T)$ and $B(T')$ is done by comparing segments of $O(x)$ bits, coding subtrees. Already compared isomorphic subtrees are removed from T and T', thus producing two new *residual* trees, T_1 and T'_1. During this process, we maintain the invariant that $T_1 = T'_1$, if and only if $T = T'$. The process ends when two distinct segments are compared, or when both residual trees are empty. In order to enforce the invariant, we use ranks of nodes in a pre-order visit. We denote by $\rho(v)$ the *rank* of node v. Two subtrees are compared by exchanging messages that contain, for each of them, the code of the subtree, the rank of the parent of the root of the subtree, and the port number at the parent leading to the subtree. If two such messages are different, the output of the comparison task is NO, and nodes of the tree corresponding to the lexicographically larger message get label 1. If the process ends with both residual trees becoming empty, the output of the comparison task is YES.

Pruning the trees by removing subtrees, instead of leaves, allows us to take advantage of the memory available at each node for sending large segments of data and thus speeding up the comparison.

Messages sent during the execution of the algorithm are of two different types. The *subtree* messages concern a single subtree, rooted at some node v. The *compound* messages combine information from different subtrees, rooted at consecutive sibling nodes. Both *compound* and *subtree messages* also contain sufficient information to obtain the rank $\rho(w)$ of the parent w of the roots of the coded subtrees, together with the port numbers leading from this node to each of the subtrees. The value of $\rho(w)$ is computed incrementally while the message climbs up the tree towards the root, and becomes the correct value of the rank when the message is sent from the root of one input tree to the root of the other.

Each node keeps three counters, whose values are bounded by n. Each node reserves one third of its memory for the counters. Hence, if $x \geq 9\lceil \log n \rceil$, counters do not overflow. We thus make the assumption that $x \geq 9\lceil \log n \rceil$ throughout the section.

Sending a message from a node to its parent requires at least two rounds. Whenever a node v wants to send a message to its parent, it sends it and keeps sending the same message in all subsequent rounds, until its parent sends a *confirmation* message back to node v. After receiving a confirmation, node v stops sending this message and may start executing some new task. This approach allows us to use memory of nodes more efficiently. In order to repeat sending the same message to its parent, node v has to keep it in its memory. However, it is more economical to keep *one* message in the memory of each child than to keep many messages coming from children in the memory of the parent, while it waits for the remaining messages to arrive.

The detailed description of the algorithm is given below.

Algorithm. `Sync Compare`
Input: two rooted trees, T and T', whose respective roots r and r' are connected by an edge e. Port numbers at r and r' corresponding to e are $d(r)$ and $d(r')$, respectively.

We first describe the fields of *subtree* messages. Let b be the code of a subtree rooted at v. A *subtree* message contains four fields.
Field 1 will eventually contain the rank $\rho(w)$ of the parent node w of node v, when the message is relayed by r (respectively r') to r' (respectively r).
Field 2 will eventually contain the port number at node w corresponding to the edge (v, w), when the message is relayed by the parent w of node v.
Field 3 contains the size s of the original subtree rooted at v.
Field 4 contains the code of the residual subtree rooted at v (hence, in general, it does not contain the code of the whole subtree rooted at v at the beginning of the execution of the algorithm).

The number of fields in messages of type *compound* may vary. Fields 1 to 3 of *compound* messages are common to all messages of this type.
Field 1 will eventualy contain the rank $\rho(v)$ of the parent v of all roots of the subtrees coded in the message. As before, this information is complete when the message is relayed by r (respectively r') to r' (respectively r).
Field 2 contains the port number, at node v, corresponding to the edge $\{v, w\}$, where w is the parent of v (and $d(v)$ if v is r or r').
Field 3 contains the position j, among all children of node v, of the root of the first subtree coded in the message (in order of increasing port numbers at v). Subsequent fields are pairs. The first element of each pair is the size s of the original subtree rooted at the corresponding child of node v; the second element of each pair contains the code of the residual subtree rooted at this child.

A further distinction among messages classifies them as *small* or *large*. A *subtree* message is *small*, if the length of the code it contains is bounded by $x/6$. A *compound* message is *small*, if the sum of the lengths of the codes it contains (further augmented by the information on original subtree sizes) is bounded by

$x/6$. All other messages are *large*. The length of the code contained in a *large subtree* message is bounded by $x/3$. Similarly, the sum of the lengths of the codes and original subtree sizes in a *large compound* message is bounded by $x/3$.

Each internal node v keeps 3 counters: counters $c(v)$ and $s(v)$ are initialized to 0; counter $p(v)$ is initialized to -1. One third of the memory of each node is reserved for these counters, while the remaining two thirds are used to memorize a message or construct a new one.

Starting from round 1 each leaf sends a *subtree* message $< -1, -1, 1, 10 >$ to its parent.

In the first round when an internal node v receives a message from all its neighbors but one, it stores the port number corresponding to the only non-transmitting neighbor in counter $p(v)$. This neighbor is the parent of v.

Consider an internal node v that already set its counter $p(v)$. Let $i_0, i_1, \ldots, i_{d(v)-2}$ be the increasing sequence of the $d(v) - 1$ integers different from $p(v)$ in $[0, \ldots, d(v) - 1]$.
At any round node v acts differently according to the following 3 possible cases:

1. messages $m_{i_0}, \ldots, m_{i_{d(v)-2}}$ from all its children are *small subtree* messages and the *subtree* message $m =< -1, -1, 1 + \sum_{j=0}^{d(v)-2} s(i_j), 1|p(v)|b(i_0) \ldots b(i_{d(v)-2})0 >$ is either *small* or *large*, where $s(i_j)$ is the size from field 3 of message m_{i_j}, and $b(i_j)$ is the code from field 4 of this message;
2. messages $m_{i_0}, \ldots, m_{i_{d(v)-2}}$ from all its children are *small subtree* messages and the *subtree* message $m =< -1, -1, 1 + \sum_{j=0}^{d(v)-2} s(i_j), 1|p(v)|b(i_0) \ldots b(i_{d(v)-2})0 >$ contains a code of length larger than $x/3$,
3. some message from its children is either *compound* or *large*.

In case 1, node v starts sending message m to its parent and sends *confirmation* messages to all its children. After receiving a confirmation from its parent, node v has sent all the information related to its subtree and will never send any other message to its parent; the whole subtree rooted at v is thus pruned from the tree.

In case 2, node v constructs the following *compound* message m.

$$m =< 1, p(v), c(v), (s(i_{c(v)}), b(i_{c(v)})), \ldots, (s(i_{c(v)+k}), b(i_{c(v)+k})) >,$$

where k is either $d(v) - c(v) - 2$ (meaning that all children have been already handled), or the maximum value such that subtree codes and sizes in m do not exceed the allowed size for *large compound* messages. Node v sends message m to its parent; in the same round, it sends confirmations to children connected through ports from $i_{c(v)}$ to $i_{c(v)+k}$ and sets counter $s(v)$ to $s(v)+\sum_{j=0}^{k} s(i_{c(v)+j})$, and counter $c(v)$ to $c(v) + k + 1$. Children that receive the confirmation have sent all the information related to their subtrees and will never send any other message to their parent. These nodes and the residual subtrees rooted at them are pruned from the tree. When node v receives a confirmation from its parent, it considers as its children only nodes corresponding to ports from $i_{c(v)}$ to $i_{d(v)-2}$ (i.e., those children that have yet to be pruned) and repeats the steps described above for case 2. When $c(v)$ reaches value $d(v)-1$, node v starts sending a *subtree*

message $< -1, -1, s(v)+1, 1|p(v)|0) >$ to its parent. This is the last message that node v will send to its parent during the execution of the algorithm.

In case 3, *compound* and *large* messages are relayed by node v to its parent. The behavior of node v is different for *compound* and for *large subtree* messages, and for *large subtree* messages it is again different depending on whether the message has been already relayed by some other node or not. Details for all possible cases are provided below.

Let i_k be the smallest port number from which v is receiving either a *large* or a *compound* message. If $c(v)$ is less than k and there exists a port i_j, for $c(v) \leq j < k$, from which v is not receiving any message in the current round, then v waits till the next round. Hence we can assume that all messages $m_{i_{c(v)}}, \ldots, m_{i_{k-1}}$, from ports $i_{c(v)}$ to i_{k-1} are *small subtree* messages. Let $s(c(v)), \ldots, s(k-1)$ be the values of fields 3 of these messages.

Let $m =< \rho, p, c, (s_1, b_1), \ldots, (s_l, b_l) >$ be a *compound* message received by node v through port i_k. Node v relays message m by sending message $< \rho + s(v) + 1 + \sum_{j=c(v)}^{k-1} s(j), p, c, (s_1, b_1), \ldots, (s_l, b_l) >$ to its parent and sending confirmation to its child connected through port i_k, without modifying any of its counters.

Let $m =< \rho, p, s, b >$ be a *large subtree* message received by node v from port i_k.

If $\rho = -1$, then message m has never been relayed by any node, and node v is the parent of the root of the subtree coded in this message. Messages from ports $i_{c(v)}$ to i_{k-1} are first used, similarly as in case 2, to send as many *compound* messages as needed. As a result, counters of v are updated and $c(v)$ becomes equal to k. Hence, node v relays message m by sending message $< 1, k, s, b >$ to its parent. It also updates its counter $s(v)$ by adding s to it, and its counter $c(v)$ by adding 1 to it. Moreover, it sends a confirmation to its child connected through port i_k. This child (and all other children connected through a port number smaller than i_k) will never send any other message to v. After receiving a confirmation from its parent, node v considers as its children only nodes connected to it through ports from $i_{c(v)}$ to $i_{d(v)-2}$, and treats subsequent messages from these children as in cases 2 or 3, depending on whether all messages are *small subtree* or not.

If $\rho \geq 1$, then message m has already been relayed by some node. Node v relays message m by sending message $< \rho + s(v) + 1 + \sum_{j=c(v)}^{k-1} s(j), p, s, b >$ to its parent, without modifying any of its counters. Node v also sends a confirmation to its child connected through port i_k. If counter $c(v)$ is still equal to 0, node v can treat subsequent messages from its children according to cases 1, 2 or 3, depending on their type. If $c(v) > 0$, then subsequent messages are handled according to cases 2 or 3 only.

Roots r and r' behave in a similar way as all the internal nodes, with the only difference that each root relays messages to the root of the other tree instead of relaying to its parent. Messages relayed in round i are also kept in memory and compared, in round $i+1$, with the corresponding message sent by the other root in round i. If the messages are different, or one of them is missing, the trees are not isomorphic and the outcome of the comparison is NO (termination messages are broadcast from each root to its respective tree; nodes in the tree whose

corresponding message is lexicographically larger, or is missing, get label 1, nodes in the other tree get label 0). The outcome of the comparison is YES, if and only if *subtree* messages are simultaneously generated by each root and match. ■

In the proof of the correctness of the algorithm, we will use the following technical lemmas.

Lemma 1. *Let m be a message to be relayed by the root of one of the input trees to the root of the other tree during execution of Algorithm* `Sync Compare`*. Let u be the parent of the roots of the trees coded in m. When m is relayed to the other root, then its field 1 contains the rank of u.*

For any instance (T, T') of the comparison problem that gives output YES, the sequences of messages sent by each root to the other are identical. Hence we can define a function $f : \mathcal{T} \longrightarrow M$, where $\mathcal{T}$ is the set of input trees and M is the set of sequences of messages, such that $f(T)$ is the sequence of messages sent from r to r' for any YES-instance (T, T') of the comparison problem.

Lemma 2. *If $f(T_1) = f(T_2)$, then T_1 and T_2 are isomorphic.*

Lemma 3. *Algorithm* `Sync Compare` *is correct.*

Proof. To prove the correctness of Algorithm `Sync Compare`, we show that its outcome is YES, if and only if input trees T and T' are isomorphic. For any deterministic algorithm, it is impossible that the roots of two isomorphic subtrees behave differently. Hence, it is straightforward that Algorithm `Sync Compare` has outcome YES *if* the input trees are isomorphic. It remains to be shown that, whenever the outcome of Algorithm `Sync Compare` is YES, then the input trees are isomorphic.

Let (T, T') be an instance of the comparison problem with output YES. Let μ_T and $\mu_{T'}$ be the sequences of messages sent by r to r' and by r' to r, respectively. Since the outcome is YES, we have $\mu_T = \mu_{T'}$. By Lemma 2, trees T and T' are isomorphic. □

We now consider the completion time of Algorithm `Sync Compare`.

Lemma 4. *Algorithm* `Sync Compare` *terminates in time $O(\max(h, n/x))$, for any pair of input trees (T, T') having at most n nodes and height bounded by h, where x is the number of memory bits at each node.*

Proof. The time needed for a node to receive at least one message from each child is linear in the height of its subtree. Indeed, nodes that receive only *small subtree* messages from their children receive a message from each of them within time linear in the height of the tallest subtree, and each node that received a message from all its children in round i, sends a message to its parent in round $i + 1$.

A node that has sent a message m to its parent in a given round keeps sending the same message in all subsequent rounds until it gets a confirmation from the parent.

Claim. A node that received a confirmation from its parent in round i, either sends a new message to its parent in round $i+2$ at the latest, or never sends a message to its parent again.

The correctness of the claim is clear in cases 1 and 2. It is also clear for a node v in case 3, if the message confirmed in round i has been created by node v and not relayed; we now prove by induction on the number of times the message has been relayed that the claim holds also if the message confirmed to v in round i was a relayed message. Hence node v still has messages to send to its parent. If node v was the first to relay the message m, then m has been generated by one of its children. If m is a *subtree* message, node v increased its counter $c(v)$ in round i and, in order to send another message to its parent, it does not need to wait for any other message from the child whose message it relayed. If m is a *compound* message, then the child w that sent it to v has already updated its counter $c(w)$, in round i, and is sending a new message, in round $i+1$, that only depends on messages sent from ports from $i_{c(w)}$ to $i_{d(w)-2}$. Node v is thus receiving input from all the children that have not been pruned yet from its subtree and is thus able to send a new message within round $i+2$ also in this case. This completes the argument for the basis of the induction. By the inductive hypothesis, if the message confirmed to a node w by its parent in round i has been relayed at most k times, then node w sends a new message within round $i+2$, provided that it still has messages to send to its parent. Let v be a node that received a confirmation from its parent in round i, for a message m relayed $k+1$ times. Let w be the child of v that sent the message m to v. For node v to get a confirmation in round i, it must have started sending message m in round $i-1$ at the latest. Hence, node w got a confirmation from its parent v in round $i-1$ at the latest. Message m has been relayed by node w for the k-th time, and hence, by the inductive hypothesis, node w is transmitting a new message to v in round $i+1$ at the latest (node w still has messages to send to its parent, as the last message sent by each node is the *subtree* message it generates). It follows that node v is receiving a message from each of the children not yet pruned from its subtree in round $i+1$ at the latest, and thus sends a new message in round $i+2$ at the latest, which completes the proof of the claim by induction.

Now consider the messages relayed by the root r of tree T to the root r' of tree T'. *Small* messages relayed by r can be only *compound*. Indeed, *subtree* messages are relayed only if they are *large*. A *small compound* message can be generated only in case 2, after generating at least one *large compound* message, or in case 3, for *small subtree* messages preceding or following a *large subtree* message that has never been relayed. Hence for each *large* message, at most two *small compound* messages can be generated. As generating a *large* message corresponds to the pruning of a subtree of size linear in x, at most $O(n/x)$ *large* messages can be generated during the execution of the algorithm. It follows that a total of at most $O(n/x)$ messages traverse edge $\{t, t'\}$ during the execution of the algorithm. The first of these messages is sent within time $\tau \in O(h)$ since the beginning of the execution. In view of the claim, after time τ edge $\{t, t'\}$ is traversed by a new message at most every two rounds. Hence, after time $O(h + n/x)$, either

a difference is found, or tree T is reduced to an empty tree and the algorithm terminates, which proves the lemma. ■

Lemmas 3 and 4 imply the following theorem.

Theorem 1. *Let $x \geq 9\lceil \log n \rceil$. There exists an automaton with x bits of memory that solves the tree comparison problem in the class of all trees of size at most n and of height at most h in time $O(\max(h, n/x))$.*

We conclude this section by establishing the following matching lower bound on the time of tree comparison with x bits of memory at nodes.

Theorem 2. *If the automaton has x bits of memory, then time $\Omega(\max(h, n/x))$ is needed to solve the tree comparison problem in the class of all trees of size at most n and of height at most h, where $h > 1$.*

4 Trade-Offs between Memory and Number of Messages

In this section we consider the asynchronous scenario and establish trade-offs between the memory size of an automaton and the number of messages needed to accomplish the tree comparison task. Consider the class of trees with at most n nodes and of maximum degree at most Δ. Suppose that the automaton $\mathcal{A}$ placed at each node of the compared trees has x bits of memory, where $x \geq c \log \Delta$, for a small constant c, whose value will be specified later.

We now describe Algorithm `Async Compare`. The algorithm is asynchronous and makes use of the codes of trees described in Section 2. Such a code can be constructed by performing a pre-order visit of a n-node tree. As opposed to the synchronous case, in this setting there is no need of complicating the code to complete the comparison fast, which required additional information to be added to the messages. Hence, we produce the code left to right by moving a token from a node to the next one in the visit by a message exchange between the owner of the token and the next node in the visit (either a child of the current owner or its parent), starting from the root. The token piggybacks a segment of the code of the input tree, of size $O(x)$, where x is the number of memory bits available at each node. When the code of a segment is long enough, it is sent up to the root of the input tree for comparison with the corresponding segment of the other input tree. The outcome of the comparison is YES, if all corresponding segments are equal. Otherwise, symmetry is broken on the basis of the first pair of different segments.

The detailed description of the algorithm is given below.

Algorithm. `Async Compare`
Input: two rooted trees, T and T', whose respective roots r and r' are connected by an edge e. Port numbers at r and r' corresponding to e are $d(r)$ and $d(r')$, respectively.

Each node v keeps 3 counters. Counter $p(v)$ is used to store the port number leading to the parent of node v. Counter $c(v)$ is used by the owner of the token to

store the port number leading to the next node, in the pre-order visit. Counter $o(v)$ is used, by each node v in the path from the root to the current owner of the token, to store the port number leading to the child whose subtree contains the owner of the token. All counters are initialized to -1; three quarters of the memory of each node are reserved to the counters, while the remaining quarter is used to store or construct a message.

At the beginning of the execution of the algorithm, roots r and r' own the token of the respective tree.

When an internal node v receives the token for the first time, it stores in $p(v)$ the port number from which it received the token. Then, v appends $|p(v)|1$ to the code segment in the token content, and sends the token through the smallest port whose number is different from $p(v)$ (this corresponds to sending the token to the first child of v). Node v also stores this port number in counter $c(v)$. If appending $|p(v)|1$ to the code segment would result in a segment that exceeds $\lfloor x/4 \rfloor$ bits, node v sends the received code segment to its parent and pauses the pre-order visit.

When a leaf v receives the token, it appends digit 0 to the code segment and sends the token back to its parent. If doing so would result in exceeding length $\lfloor x/4 \rfloor$ of the segment, v pauses the pre-order visit and sends the code segment back to its parent.

An internal node v that receives the code segment from one of its children, updates the value of counter $c(v)$ to the port leading to the next child, or to -1, if all children have been already visited. Then, v sends the token to its next child, appending digit 1 to the code segment, or sends it back to its parent (if all children have been already visited), appending digit 0. Similarly as before, the visit is paused and the code segment is sent up unaltered, if appending would result in a too large segment.

Code segments are sent up the tree to the root as follows. When a node v (including roots of the input trees) gets a code segment from a port $i \neq p(v)$, it stores value i in counter $o(v)$ and sends the code segment to its parent (or to the root of the other tree in the case when v is the root of one of the input trees).

The root of an input tree can receive a code segment either from one of its children, or from the root of the other tree. The code segment that arrives first is stored in the memory (when code segments from the other root and a child arrive at the same time, the one coming from the child is stored), and compared with the corresponding one as soon as it is received.

After comparison of two corresponding code segments, if no difference is found, roots r and r' send a request for the next code segment, through port $o(r) = o(r')$. The request is forwarded by each node v in the path connecting the root to the current owner of the token of its tree, by sending it through port $o(v)$.

If a difference in two corresponding segments exchanged by the roots r and r' is found, or the token got back to one root from its last child while the other root has still a segment to compare, the trees are not isomorphic and the outcome of the comparison is NO. The root that sent the lexicographically larger segment (or that received the token back first), gets labels 1, while the other root gets

label 0. The outcome is then broadcast to all nodes in the trees, starting from the roots; each node in the tree gets the label assigned to the root of the tree it belongs to.

It remains to be described how the visit is restarted by the current owner of the token. When the current owner v of the token receives a request for the next segment, it sends the token to the next node in the pre-order visit as follows. If v is an internal node whose counter $c(v)$ points to the first child, v sends the token containing code $|p(v)|1$ to this child. If v is an internal node whose counter $c(v)$ points to a subsequent child, v sends the token containing digit 1 to this child. If v is either a leaf or an internal node whose counter $c(v) = -1$, it sends the token containing digit 0 to its parent.

The outcome of the comparison is YES, if all compared code segments coincide, and both roots get the token back together with corresponding code segments. This outcome is broadcast to all nodes of the trees. ■

Theorem 3. *There exists an automaton with x bits of memory that solves the tree comparison problem in the class of all tress of size at most n, using $O(n^2/x)$ messages, for $4\lceil \log n \rceil \leq x \leq n$.*

Proof. The algorithm uses 3 counters whose values are bounded by n. We reserve 3/4 of the memory for the counters and the rest to store or produce a message. When the memory available at each node is at least $4\lceil \log n \rceil$, the memory reserved to each counter is large enough to avoid overflows, and the correctness of Algorithm `Async Compare` follows from the fact that two trees are isomorphic, if and only if they have the same code.

As for the number of messages sent during the execution of the algorithm, performing the visit requires a message for each edge traversed by the token, thus totalling $2n - 2$ messages when the token gets back to the root. Sending each code segment to the root of the input tree and resuming the visit requires 2 messages for each traversed edge (one for sending the code segment up, and one for sending the request for the next code segment down). As $O(n/x)$ code segments are produced during the execution of the algorithm and less than n edges are traversed by each code segment, the total number of messages sent for delivering code segments to the root and resuming the visit is $O(n^2/x)$. $O(n/x)$ messages are exchanged between the roots of the input trees. Finally, broadcasting the outcome of the comparison requires at most $n - 1$ additional messages for each input tree, which completes the proof. □

We conclude by establishing the following matching lower bound on message complexity.

Theorem 4. *If the automaton has x bits of memory, then $\Omega(n^2/x)$ messages are needed to solve the tree comparison problem on the class of all trees of size at most n.*

Proof. Let $k = \lfloor n/3 \rfloor - 1$. For simplicity assume that x divides k (modifications in the general case are straightforward). Consider the following class $\mathcal{C}$ of trees

of size at most n. A tree of class $\mathcal{C}$ is rooted at node $r = v_1$ and has a branch $(v_1, \ldots, v_k, w_1, \ldots, w_{k+1})$. Ports at every node of this branch are: 0 corresponding to the edge joining the node with its parent, and 1 corresponding to the edge joining the node with its child. Moreover, each of the nodes w_i, for $1 = 1, \ldots, k$, may or may not have another child w'_{i+1} with port number 2 at w_i corresponding to the joining edge. All nodes w'_i are leaves. Thus there are 2^k trees in class $\mathcal{C}$.

Suppose that some automaton $\mathcal{A}$ with x bits of memory solves the comparison problem for the class $\mathcal{C}$ of trees. Consider the set Σ of symmetric instances of the comparison problem for the class $\mathcal{C}$, i.e., instances where trees T and T' are isomorphic and belong to $\mathcal{C}$, and such that trees in distinct instances are not isomorphic. There are 2^k instances in Σ. For any instance $\sigma \in \Sigma$ and any $i = 1, \ldots, k-1$, let $v_i(\sigma)$ and $\overline{v_i}(\sigma)$ be the two nodes at distance i from the roots of the compared trees. Let $g_i(\sigma)$ denote the sequence of messages sent by each of these two nodes to its parent during a synchronous execution of the comparison task with automaton $\mathcal{A}$ on instance σ.

Claim. For any $i = 1, \ldots, k-1$ and any distinct instances σ, σ' from Σ, we have $g_i(\sigma) \neq g_i(\sigma')$.

To prove the claim, suppose that for some $i = 1, \ldots, k-1$ and some distinct instances σ, σ' from Σ, we have $g_i(\sigma) = g_i(\sigma')$. Thus in every round of a synchronous execution on instance σ, the states of all nodes $v_j(\sigma)$ and $\overline{v_j}(\sigma)$, for $j < i$, are the same as the states of the respective nodes in a synchronous execution on instance σ'. Consider an instance τ of the comparison problem in which one of the compared trees comes from the instance σ and the other from the instance σ'. In each round of a synchronous execution on instance τ, the states of all nodes $v_j(\tau)$ and $\overline{v_j}(\tau)$, for $j < i$ are the same as the states of the respective nodes in a synchronous execution on instance σ. This leads to a contradiction, as on the instance σ all nodes must enter a state from the set YES and on instance τ all nodes must enter a state from the set $NO_0 \cup NO_1$. This proves the claim.

Let M_s denote the number of distinct sequences of at most s messages that can be sent by a node on port 0. Since there are $y = 2^x$ possible states of a node, we have $M_s = \sum_{i=0}^{s} y^i = \frac{y^{s+1}-1}{y-1}$. Moreover, if $sx \leq j$, then $y^{s+1} - 1 = 2^{xs} \cdot 2^x - 1 \leq 2^j \cdot 2^x - 1 \leq 2^{j+1} \cdot (2^x - 1) = 2^{j+1} \cdot (y-1)$. This implies that, for any $j = 1, \ldots, k-2$, if $sx \leq j$, then $M_s \leq 2^{j+1}$.

For any instance $\sigma \in \Sigma$ and any $i = 1, \ldots, k-1$, let $|g_i(\sigma)|$ denote the number of messages in $g_i(\sigma)$. The above estimate on M_s and the claim imply that, in any set of at least 2^{j+2} instances from Σ, and for any $i = 1, \ldots, k-1$, there is a subset of at least 2^{j+1} instances σ, such that $|g_i(\sigma)| \geq j/x$. Thus we can construct a descending sequence of sets of instances $\Sigma \supset \Sigma_1 \supset \Sigma_2 \supset \cdots \supset \Sigma_{k-2}$, such that Σ_i has at least 2^{k-i} elements and, for all $\sigma \in \Sigma_i$, we have $|g_i(\sigma)| \geq (k-i-1)/x$. This implies that, for any $\sigma \in \Sigma_{k-2}$, the number of messages sent in a synchronous execution on instance σ is at least $(1/x)(1+2+\ldots(k-2)) \in \Omega(k^2/x) = \Omega(n^2/x)$. ■

References

1. Attiya, H., Snir, M., Warmuth, M.: Computing on an Anonymous Ring. Journal of the ACM 35, 845–875 (1988)
2. Attiya, H., Snir, M.: Better Computing on the Anonymous Ring. Journal of Algorithms 12, 204–238 (1991)
3. Boldi, P., Vigna, S.: Computing anonymously with arbitrary knowledge. In: Proc. 18th ACM Symp. on Principles of Distributed Computing, pp. 181–188 (1999)
4. Burns, J.E.: A formal model for message passing systems, Tech. Report TR-91, Computer Science Department, Indiana University, Bloomington (September 1980)
5. Codenotti, B., Gemmell, P., Simon, J.: Symmetry breaking in anonymous networks: characterizations. In: Proc. 4th Israel Symposium on Theory of Computing and Systems (ISTCS 1996), pp. 16–26 (1996)
6. Diks, K., Kranakis, E., Malinowski, A., Pelc, A.: Anonymous wireless rings. Theoretical Computer Science 145, 95–109 (1995)
7. Fredrickson, G.N., Lynch, N.A.: Electing a leader in a synchronous ring. Journal of the ACM 34, 98–115 (1987)
8. Hirschberg, D.S., Sinclair, J.B.: Decentralized extrema-finding in circular configurations of processes. Communications of the ACM 23, 627–628 (1980)
9. Jurdzinski, T., Kutylowski, M., Zatopianski, J.: Efficient algorithms for leader election in radio networks. In: Proc. 21st ACM Symp. on Principles of Distr. Comp. (PODC 2002), pp. 51–57 (2002)
10. Kowalski, D., Pelc, A.: Leader election in ad hoc radio networks: a keen ear helps. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 521–533. Springer, Heidelberg (2009)
11. Kranakis, E.: Symmetry and Computability in Anonymous Networks: A Brief Survey. In: Proc. 3rd Int. Conf. on Structural Inform. and Comm. Complexity, pp. 1–16 (1997)
12. Kranakis, E., Krizanc, D., van der Berg, J.: Computing Boolean Functions on Anonymous Networks. Information and Computation 114, 214–236 (1994)
13. Lindell, S.: A logspace algorithm for tree canonization. In: Proc. 24th ACM Symposium on Theory of Computing (STOC 1992), pp. 400–404 (1992)
14. Nakano, K., Olariu, S.: Uniform leader election protocols for radio networks. IEEE Trans. on Parallel Distributed Systems 13, 516–526 (2002)
15. Peterson, G.L.: An $O(n \log n)$ unidirectional distributed algorithm for the circular extrema problem. ACM Trans. on Prog. Languages and Syst. 4, 758–762 (1982)
16. Sakamoto, N.: Comparison of Initial Conditions for Distributed Algorithms on Anonymous Networks. In: Proc. 18th ACM Symp. on Principles of Distributed Computing (PODC 1999), pp. 173–179 (1999)
17. Willard, D.E.: Log-logarithmic selection resolution protocols in a multiple access channel. SIAM J. on Computing 15, 468–477 (1986)
18. Yamashita, M., Kameda, T.: Computing on anonymous networks. In: Proc. 7th ACM Symp. on Principles of Distributed Computing (PODC 1988), pp. 117–130 (1988)
19. Yamashita, M., Kameda, T.: Electing a leader when procesor identity numbers are not distinct. In: Bermond, J.-C., Raynal, M. (eds.) WDAG 1989. LNCS, vol. 392. Springer, Heidelberg (1989)
20. Yamashita, M., Kameda, T.: Computing on anonymous networks: Part I - characterizing the solvable cases. IEEE Trans. Parallel and Distributed Systems 7, 69–89 (1996)

Periodic Data Retrieval Problem in Rings Containing a Malicious Host*
(Extended Abstract)

Rastislav Královič and Stanislav Miklík

Dept. of Computer Science
Comenius University
Bratislava, Slovakia

Abstract. In the problems of *exploration of faulty graphs*, a team of cooperating agents is considered moving in a network containing one or more nodes that can harm the agents. A most notable among these problems is the problem of *black hole location*, where the network contains one node that destroys any incoming agent, and the task of the agents is to determine the location of this node. The main complexity measure is the number of agents needed to solve the problem. In this paper we begin with a study of malicious hosts with more varied behavior. We study the problem of *periodic data retrieval* which is equivalent to *periodic exploration* in fault-free networks, and to *black hole location* in networks with one black hole. The main result of the paper states that, in case of rings, it is sufficient to protect the internal state of the agent (i.e. the malicious host cannot change or create the content of agent's memory), and the periodic data retrieval problem is solvable by a constant number of agents.

Keywords: malicious host, gray hole, periodic data retrieval.

1 Introduction

Problems related to the exploration of faulty environments have received much attention over the recent years, with motivation coming from the areas like distributed mobile computing, graph exploration, and robotics. *Distributed mobile computing* is a software engineering paradigm for designing distributed systems, in which, instead of stationary processes exchanging data over the network, the data are located in the nodes of the network (*hosts*), and pieces of software (*agents*) are sent over the network to perform computation. A host is capable of receiving an agent, preparing a workspace for it, running it using the host's resources, and, upon request, serializing it and sending its complete state over the network to another host. This way of computation has many advantages over the classical paradigm, but also presents new problems, mainly concerning the security of both agents and hosts [6,25,30,33]. While many standard techniques to

* This research was supported by grant APVV 0433-06.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 157–167, 2010.

protect the host from possibly harmful agents can be used (e.g. sandboxing), protecting the agent from a possibly malicious host is much harder [27,28,34,32,35], since the host provides the whole runtime environment for the agent.

In the *graph exploration* problems (e.g. [7,20,23]), the resources needed to navigate in a graph are studied. A number of computing devices (automata) are moving in a graph, with a common task to perform (e.g. traversing every vertex in the graph, meeting in a common vertex, visiting every vertex infinitely many times, etc). The resources (mainly the memory requirements of the devices) needed for the solvability of the given problem are the main focus of study. The overall number of moves is often considered, as well. Apart from practical applications (e.g. exploring unknown environments by mobile robots), results in these problems have often profound theoretical consequences. Perhaps the oldest problem from this class is the problem of graph exploration by a finite automaton (see e.g. [5,8,19,26] and references therein) where a single agent is modeled by a finite state machine, and the goal is to traverse the whole graph. Numerous other problems and variants have been considered for teams of agents (e.g. [1,2,3,4,11,18,21,22,24,29,31]).

The problems of *exploration of faulty graphs* may be viewed as a theoretical tool for solving the problems arising from the mobile computing area, and at the same time as a natural generalization of the graph exploration problems. A team of cooperating agents moving in a graph is considered. The graph contains (one or multiple) malicious vertices that have the ability to harm the agents in some way. The goal of the agents is to perform some task in spite of the presence of the faulty nodes. The most studied in this context is the *black hole location* (e.g. [13,14,12,15,17]) problem where the graph contains one special node called *black hole* that destroys any incoming agent without any observable trace. The task of the agents is to determine the location of the black hole.

The main resource to be optimized is the amount of network traffic needed to solve the task. A notable issue in this setting is that the agents are usually not allowed to duplicate, and the complexity is then measured primarily by the number of agents, and secondarily by the overall number of performed moves. While this limitation is natural when thinking about applications like terrain exploration by physical devices, in the context of software agents the duplication is a widely used technique. To illustrate the effect of agent duplication, consider the black hole location problem in a directed graph[1]. With the agent duplication, a simple algorithm can solve the problem with the number of moves equal the number of arcs: if an agent arrives to an already visited vertex, it dies; otherwise, it sends its copy along all outgoing arcs. Obviously, every arc of the graph is visited exactly once. However, as shown in [10], without multiplication $\Omega(2^n)$ moves may be needed in the worst case, where n is the number of vertices. In this paper we provide efficient algorithms for our problem, that do not rely on agent multiplication.

[1] With the requirement that after extracting the black hole, the rest of the graph is strongly connected.

While studying the black hole location problem, much attention has been devoted to the impact of different properties of the system to the complexity of the solution. The properties in question are e.g. the existence of a common homebase for the agents, mode of communication (pebbles, whiteboards, face-to-face), synchrony etc. We adopted the widely studied model of asynchronous agents sharing a common homebase, and communicating via shared storage (aka *whiteboards*).

As already mentioned, black hole is a particular type of a malicious host with a very simple behavior: killing every agent instantly. In reality, a host has many ways to harm the agents: it may not only kill any agent residing in it at any time, it may also duplicate agents, introduce fake agents, tamper the runtime environment (e.g. changing the contents of the whiteboard), or disobey communication protocols (e.g. do not execute agents in FIFO order). This paper is the first attempt to study how the abilities of a malicious host affect the solvability of exploration problems.

It is easy to see that in an asynchronous setting it is not possible to locate the malicious host even if it is a *gray hole* (i.e. a host which only capability is to kill the agent at any time, but otherwise provides reliable runtime environment and follows the communication protocol). Indeed, consider a case when links to a vertex v are slow, and the gray hole located in $w \neq v$ does not kill any agent during the first phase of the computation. Since the location problem requires agents to decide, after finite time, on the location of the malicious host, they must assume that the gray hole is v.

In our contribution we focus on a modification of the location problem called *periodic data retrieval*. Let us suppose that in every non-faulty vertex, an infinite sequence of data items is generated that have to be gathered in the homebase, a scenario that is typical e.g. in sensor networks. The aim is to design an agent-based protocol that visits every non-faulty vertex infinitely many times[2], and reports the data to the homebase. We are interested in the number of agents sufficient to solve the problem in a network with a malicious host of a given type. In reliable networks, one agent is sufficient to solve the problem, and the problem reduces to the *periodic exploration* problem studied e.g. in [16,9] and references therein, where the aim was to minimize the number of moves between two consecutive visits of a vertex. If the malicious host is a black hole, then the number of agents for the location and periodic retrieval problems is the same: once the location of the black hole is known to the surviving agent, it starts to periodically traverse the non-faulty nodes. On the other hand, in a solution to the periodic data retrieval problem, the location of the black hole is known in the homebase after the first data item from every non-faulty vertex is retrieved.

In an approach similar to the research on the black hole search problem[13], we first analyze the case of ring networks, with the hope to develop techniques that could be later useful in the general case.

[2] Throughout the whole study we rely on a fair scheduler, so we disregard possible liveness issues.

It is easy to see that $n-1$ agents are sufficient to solve the periodic data retrieval problem in any 2-connected network with n vertices and known topology, regardless of the abilities of the malicious host: each of the agents guesses one of the $n-1$ possible locations of the malicious host, and periodically traverses the rest of the graph. Our main result shows that, in the case of rings, it is sufficient to protect the internal state of the agent (i.e. the malicious host cannot change or create the content of agent's memory), and the periodic data retrieval problem is solvable by a constant number of agents.

The rest of the paper is organized as follows. In Section 2, we give a more formal definition of the model. In Section 3 we prove the main result for the special case of a gray hole. In Section 4 we discuss the solution of the general case. Due to space constraints, the formal treatment of the general case has been deferred to the Appendix.

2 Preliminaries

2.1 Model of the System

An *agent* is an independent computational process with unique ID, internal memory, and an interface to a shared memory provided by a *host*. A *host* is a device capable of running agents, equipped with shared storage (called *whiteboard*), and with labeled communication ports. The host contains an order-preserving queue of incoming agents for each port, and a queue of *sleeping* agents. The network consists of a number of hosts connected by bidirectional asynchronous FIFO links forming an undirected graph[3] G. The agents share a common homebase, and have distinct identifiers. Every host as its unique identifier, and the complete map (i.e. G, port labels, and host identifiers) is known to the agents.

In a proper operation, serialized agents may arrive to a host on any of its links, they are stored in the appropriate incoming queues in the order of their arrival. Agents may access the shared memory (called *whiteboard*) using a fair locking mechanism. Agents may be waiting for some condition to become true; these are stored in the queue of sleeping agents. An agent may decide to leave the host, in which case it is serialized, and stored in the outgoing queue of the corresponding link. It is important to note that all the queues are order-preserving, and that when an agent holding the lock for the whiteboard is about to leave, it is stored in the queue before the lock is released; i.e. it is possible for the agent to update the whiteboard and leave in one atomic step. All our algorithms work in a serialized fashion: the incoming agent requests the whiteboard lock[4], and after succeeding it performs all its actions until it either leaves the host(and the lock is released by the host afterwards), or goes to sleep. When awaken from sleep (because the condition it had been waiting for became true), it again performs all its actions while holding the lock.

A host that deviates from the proper operation in any way is called *malicious*, non-faulty hosts are called *honest*.

[3] We shall interchangeably refer to the vertices of G as *vertices, nodes* or *hosts*.
[4] The lock is presented in a FIFO manner.

2.2 Problem Definition

In the *periodic data retrieval* problem, the aim is to deliver the data from any honest node v to the homebase infinitely many times. During the delivery, the data may be stored in an intermediate node, and possibly read by another agent before finally reaching the homebase.

Definition 1. *Consider an arbitrary execution of the system. A node w is said to be* reported from time t *if there is a sequence of (not necessarily distinct) agents $A_0, \ldots, A_r$, a sequence of nodes $v_0 = w, v_1, \ldots, v_r$ for some r, such that v_r is the homebase, and an increasing sequence of times $t \leq t_0 < t_1 < \cdots < r_r$ such that A_i visits v_i at time t_i, and v_{i+1} at time t'_i, where $t_i < t'_i < t_{i+1}$.*

An algorithm solves the periodic data retrieval *problem, if, in any execution, for any time t, and any node v, v is reported from time t.*

2.3 Malicious Host

In this work we suppose that the system is a ring of n nodes, and contains one malicious host (denoted ω) that can behave in an arbitrary way, except that it can not change the internal state of an agent (i.e. contents of its local variables)[5], or create an agent with a given state. Hence, the only way ω may manipulate with the agents is to store and copy agents that have entered it. Let us now present some generic techniques to restrict the possible behavior of ω.

Lemma 1. *W.l.o.g. ω does not*

1. *forward an incoming agent without running it,*
2. *send an agent A without A's request, or*
3. *forward A more than once.*

Proof: Each agent A has an internal variable `transfer_ID` for the ID of the host it wants to be transfered to. Upon entering a node, A first check the ID of the current host against `transfer_ID`; if it is not equal, A dies. Before A requests to be transfered, it sets `transfer_ID` accordingly. Hence, forwarding an agent without running it, or sending it over a link without request is equivalent to killing it.

Since ω can only send A over a link it requested, the only way ω can forward A more than once is to store it in a state in which it requests transfer, and send in more than once over the same link. However, A can maintain an internal variable `steps` counting its number of moves[6]. Upon entering a node, it increments `steps`, and stores it in the node together with its ID. Hence, A can detect if it was forwarded over a link more than once, and die in that case. □

[5] Since the code of the agents is identical and publicly known, it is easy to prevent ω from changing the code by killing any altered agent entering any honest node.

[6] We consider this to be an unbounded integer, but it can be made bounded using proper modulus.

3 Solution with Reliable Whiteboards

In this section we consider, for the clarity of exposition, a simple special case of the malicious host called *gray hole*. The gray hole's only ability is to kill any agent anytime. If the agent has acquired the whiteboard lock, the lock is released. Apart from this, the gray hole provides reliable running environment. Let us start with a simple lower bound:

Lemma 2. *Two agents are not sufficient to solve the periodic data retrieval problem on a ring with one malicious host ω, even if ω is a gray hole.*

Proof: Assume that there is a protocol solving the periodical data retrieval problem that with two agents, A and B. Clearly, A and B must, at the beginning, leave the homebase in opposite directions: the first agent who leaves the homebase can be killed immediately by ω, so the second one cannot wait and has to leave in the opposite direction. Let v denote the neighbor of the homebase to which B left, w_1 the other neighbor, and w_2 the neighbor of w_1.

Consider an execution γ_1, in which ω is located in v, and B is killed on its first move; A must visit every honest node infinitely many times. Let us take another execution, γ_2, in which B is not killed but slow. A cannot distinguish γ_1 and γ_2, and has to traverse all vertices except v. Let us suppose that in γ_2, ω is located in w_1, and A is killed immediately before it left w_1 to w_2 for the first time. Clearly, B has to visit w_2 from the opposite direction. Finally consider an execution γ_3 in which ω is located in w_2, and A is killed just after entering it for the first time. B cannot distinguish γ_2 and γ_3, enters w_2 from the opposite direction, and is killed. □

For the case of a gray hole, in addition to Lemma 1, the following normal form can be considered:

Lemma 3. *W.l.o.g. ω does not*

1. *break the FIFO property, or*
2. *kill an agent in the middle of its computation.*

Proof: In every node, the list of outgoing and incoming agents is stored. When an agent A is about to leave a node (while holding the lock), it reads the list of outgoing agents (containing all agents that left to the port before), and adds its ID to the list. Upon arrival, it checks the list of incoming agents. If a difference is found, A dies; otherwise, its ID is added to the end of the list. Hence, breaking the FIFO property is equivalent to killing the agent.

Moreover, suppose an agent A is killed in the progress of its computation, and later an agent B enters ω. Since A, as its first action put its ID on the list of incoming agents, B starts waiting until A's ID appears on a list of outgoing agents. Since A has been killed, B will be waiting forever, effectively being killed, too. □

From now on we shall consider the execution fulfilling the assertions of Lemmas 1 and 3, i.e. ω kills agents only either just after entering (in which case we shall

say that it was killed on the incoming link), or just before leaving (in which case we shall say it was killed on the outgoing link). The algorithm uses a well known technique of *cautious walk*: before entering a link, an agent A marks it with a "danger" flag with its ID. If A manages to travel there and back along the link, it removes the flag, and moves to the neighboring vertex.

The whole algorithm works as follows: every agent uses the cautious walk to proceed until it finds a link that is marked with a "danger" flag. In this case it bounces and continues (using cautious walk) in the opposite direction. If a "danger" flag is set in the opposite direction, too, A waits until at least one link becomes free. In order to prove its correctness, we first show that

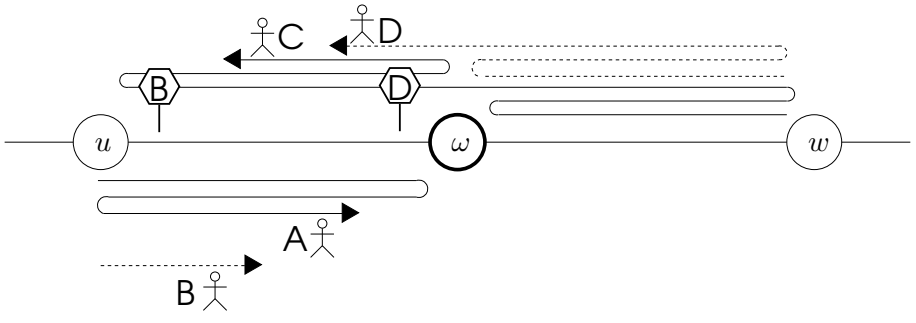


Fig. 1. At most 4 agents die on a link

Lemma 4. *During the exexution of the algorithm at most 8 agents die in* ω.

Proof: Consider the gray hole ω, and its neighbors u, w (Figure 1). First consider agents making their cautious step from u to ω. Let A be first such agent that is killed. If there is A's flag in u at that time, no other agent performs a cautious step from u to ω. If there is no A's flag in u, the next agent starting its cautious step, B, sets its flag in u. However, due to 3, B is killed, and its flag remains in u. A symmetric reasoning for agents going from ω to u, and for the other neighbor gives at most 8 agents killed in ω. □

Theorem 1. *9 agents are sufficient to solve the periodic data retrieval problem.*

Proof: Due to Lemma 4, if the algorithm starts with 9 agents, at least one agent survives. What remains to be shown is that every honest vertex is reported to the homebase infinitely many times. First let us argue that every honest vertex is visited infinitely often. Let us consider, for the sake of contradiction, a maximal continuous area $\mathcal{A}$ consisting of honest vertices that are visited only finitely many times. Let u and v be the two vertices delimiting $\mathcal{A}$. Clearly, at least one of them is not ω (since there is at least one agent alive), so is visited infinitely often. W.l.o.g. let this vertex be u. Consider a time t after which no vertex from $\mathcal{A}$ is visited. An agent comes to u after t from outside $\mathcal{A}$. Since it does not continue to enter $\mathcal{A}$, there must be a flag in u. The flag was raised by an agent B going

into the first vertex of $\mathcal{A}$. However, vertices of $\mathcal{A}$ are honest, so B would not die before removing the flag. Hence, B is still in $\mathcal{A}$: a contradiction.

The fact that every vertex is actually reported comes as a simple consequence of the fact that agents bounce only when they see a flag of another agent. If the agents write to the whiteboard the observed state of all vertices they visited, this information can be eventually propagated to the homebase. □

4 General Solution

In the previous section we showed how to solve the periodic data retrieval problem if the malicious host provides reliable storage. Now let us consider a situation when the malicious host can change the contents of the whiteboard at will. In general it means that each agent entering ω is provided with a different local view of the whiteboard. Hence, it is not possible to use the techniques from the previous section to detect the malicious activities. In order to overcome this difficulty we replace the cautious walk with a *double step*: a pattern of two moves forward, on move back, one move forward, and two moves back. An agent performing a double step scans three consecutive vertices, and by storing proper information in the vertices it is possible to detect any malicious activity (killing an agent, breaking the FIFO, changing the whiteboard, etc) of the middle vertex by comparing the information in all three vertices.

However, a number of new problems arise. First, even if some malicious activity is detected, it is not clear which of the two consecutive vertices is ω. It would be tempting to argue that when ω is located with the accuracy of two vertices, two special agents are waken up in the homebase, and each of them starts to traverse the circle with the exception of one vertex. The problem with this approach is that it is not possible to distinguish a dead agent from a slow one, and the checking procedure might have been started by a "false alarm" leading to the loss of the checking agents. To overcome this problem, the agents in the algorithm start start in an *exploring mode* in which they use the double steps to explore the ring in one direction; upon finding a place with a potential ω (i.e. a vertex with a "danger" flag set by some agent), the first two of the incoming agents switch to a *checking mode*, in which they travel in the opposite direction, and each of them evades one of the two possibly dangerous vertices. However, in doing so, they use the double steps again (with the exception of the destination where they may enter a possibly dangerous vertex). If a checking agent successfully checks its destination, it switches back to the exploration mode. It switches back to exploring mode also when it realizes that the flag that caused it to enter the checking mode was deleted.

If an exploring agent arrives to a vertex with a flag from which already two checking agents departed, it would wait there. However, the node in which it is waiting could be ω, and after a number of waiting agents being collected there, all of them could be killed. To avoid this, a mechanism is devised for the agents not to wait in the same vertex. This involves the agents' setting a flag in the previous vertex, and periodically checking if the original flag has been released.

The role of the periodic checking is to spread the information about the deletion of the flag.

The whole algorithm is complicated by the fact that the agents can not rely on a value read from a single whiteboard. Hence they use a protocol for reading and writing that always requires to access at least two vertices. In addition to this, care must be taken to the handling of waiting agents, and agents changing their direction.

With all the technical issues solved, we are able to proof the following theorem:

Theorem 2. *The periodic data retrieval problem can be solved on a ring by 27 agents.*

5 Conclusion

We studied, for the first time a generalization of the black hole search problem to more powerful kinds of malicious hosts. We provided an algorithm to solve the periodic data retrieval problem on rings with constant number of agents in the case that the malicious host cannot change or create an agent's state. A number of questions remain open, mainly:

1. Provide a solution for general graphs.
2. Provide a solution in case of unknown topology.
3. Provide a solution in case the agent's state can be changed by ω.
4. Develop non-trivial lower bounds.

References

1. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. Kluwer, Dordrecht (2003)
2. Arkin, E., Bender, M., Fekete, S., Mitchell, J.: The freeze-tag problem: how to wake up a swarm of robots. 13^{th} ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), 568–577 (2002)
3. Barriere, L., Flocchini, P., Fraigniaud, P., Santoro, N.: Capture of an intruder by mobile agents. In: Proc. 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA 2002), pp. 200–209 (2002)
4. Bender, M., Slonim, D.: The power of team exploration: Two robots can learn unlabeled directed graphs. In: 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), pp. 75–85 (1994)
5. Budach, L.: On the solution of the labyrinth problem for finite automata. Elektronische Informationsverarbeitung und Kybernetic 11, 661–672 (1975)
6. Chess, D.M.: Security issues in mobile code systems. In: Vigna, G. (ed.) Mobile Agents and Security. LNCS, vol. 1419, pp. 1–14. Springer, Heidelberg (1998)
7. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. ACM Transactions on Algorithms 4(4) (2008)
8. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. ACM Transactions on Algorithms 4(4) (2008)

9. Czyzowicz, J., Dobrev, S., Gasieniec, L., Ilcinkas, D., Jansson, J., Klasing, R., Lignos, Y., Martin, R.A., Sadakane, K., Sung, W.-K.: More efficient periodic traversal in anonymous undirected graphs. CoRR abs/0905.1737 (2009)
10. Czyzowicz, J., Dobrev, S., Královič, R., Miklík, S., Pardubská, D.: Black hole search in directed graphs. In: Kutten, S. (ed.) SIROCCO 2009. LNCS, vol. 5869, pp. 126–140. Springer, Heidelberg (2009)
11. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. In: IEEE (ed.) Proceedings: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, October 22-24, vol. 1, pp. 355–361 (1990); Formerly called the Annual Symposium on Switching and Automata Theory. IEEE catalog number 90CH29256. Computer Society order no. 2082
12. Dobrev, S., Flocchini, P., Kralovic, R., Santoro, N.: Exploring an unknown graph to locate a black hole using tokens. In: IFIP TCS. IFIP, vol. 209, pp. 131–150. Springer, Heidelberg (2006)
13. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 166–179. Springer, Heidelberg (2001)
14. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Finding a black hole in an arbitrary network: optimal mobile agents protocols. In: Proc. of 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pp. 153–162 (2002)
15. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. Algorithmica 48(1), 67–90 (2007)
16. Dobrev, S., Jansson, J., Sadakane, K., Sung, W.-K.: Finding short right-hand-on-the-wall walks in graphs. In: Pelc, A., Raynal, M. (eds.) SIROCCO 2005. LNCS, vol. 3499, pp. 127–139. Springer, Heidelberg (2005)
17. Dobrev, S., Santoro, N., Shi, W.: Locating a black hole in an un-oriented ring using tokens: The case of scattered agents. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 608–617. Springer, Heidelberg (2007)
18. Fraigniaud, P., Gasieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks 48(3), 166–177 (2006)
19. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. 345(2-3), 331–344 (2005)
20. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Impact of memory size on graph exploration capability. Discrete Applied Mathematics 156(12), 2310–2319 (2008)
21. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Impact of memory size on graph exploration capability. Discrete Applied Mathematics 156(12), 2310–2319 (2008)
22. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Tree exploration with advice. Inf. Comput. 206(11), 1276–1287 (2008)
23. Fraigniaud, P., Ilcinkas, D., Rajsbaum, S., Tixeuil, S.: The reduced automata technique for graph exploration space lower bounds. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) Theoretical Computer Science. LNCS, vol. 3895, pp. 1–26. Springer, Heidelberg (2006)
24. Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: Bansal, N., Pruhs, K., Stein, C. (eds.) SODA, pp. 585–594. SIAM, Philadelphia (2007)
25. Greenberg, M., Byington, J., Harper, D.G.: Mobile agents and security. IEEE Commun. Mag. 36(7), 76–85 (1998)
26. Hoffmann, F.: One pebble does not suffice to search plane labyrinths. In: Gecseg, F. (ed.) FCT 1981. LNCS, vol. 117, pp. 433–444. Springer, Heidelberg (1981)

27. Hohl, F.: Time limited blackbox security: Protecting mobile agents from malicious hosts. In: Vigna, G. (ed.) Mobile Agents and Security. LNCS, vol. 1419, pp. 92–113. Springer, Heidelberg (1998)
28. Hohl, F.: A framework to protect mobile agents by using reference states. In: Proc. of the 20th Int. Conf. on Distributed Computing Systems, ICDCS 2000 (2000)
29. Marco, G.D., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 271–282. Springer, Heidelberg (2005)
30. Oppliger, R.: Security issues related to mobile code and agent-based systems. Computer Communications 22(12), 1165–1170 (1999)
31. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. J. Algorithms 33, 281–295 (1999)
32. Sander, T., Tschudin, C.F.: Protecting mobile agents against malicious hosts. In: Vigna, G. (ed.) Mobile Agents and Security. LNCS, vol. 1419, pp. 44–60. Springer, Heidelberg (1998)
33. Schelderup, K., Ones, J.: Mobile agent security - issues and directions. In: Zuidweg, H., Campolargo, M., Delgado, J., Mullery, A. (eds.) IS&N 1999. LNCS, vol. 1597, pp. 155–167. Springer, Heidelberg (1999)
34. Ng, S.K., Cheung, K.: Protecting mobile agents against malicious hosts by intention spreading. In: Proc. 1999 Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 1999), pp. 725–729 (1999)
35. Vitek, J., Castagna, G.: Mobile computations and hostile hosts. In: Tsichritzis, D. (ed.) Mobile Objects, pp. 241–261. University of Geneva (1999)

A Continuous, Local Strategy for Constructing a Short Chain of Mobile Robots*

Bastian Degener, Barbara Kempkes, Peter Kling, and Friedhelm Meyer auf der Heide

Heinz Nixdorf Institute
Computer Science Department
University of Paderborn
{degener,barbaras,kronos,fmadh}@uni-paderborn.de

Abstract. We are given an arbitrarily shaped chain of n robots with fixed end points in the plane. We assume that each robot can only see its two neighbors in the chain, which have to be within its viewing range. The goal is to move the robots to the straight line between the end points. Each robot has to base the decision where to move on the relative positions of its neighbors only. Such *local* strategies considered until now are based on discrete rounds, where a round consists of a movement of each robot. In this paper, we initiate the study of continuous local strategies: The robots may perpetually observe the relative positions of their neighbors, and may perpetually adjust their speed and direction in response to these observations. We assume a speed limit for the robots, that we normalize to one, which corresponds to the viewing range. Our contribution is a continuous, local strategy that needs time $\mathcal{O}(min\{n,(OPT+d)\log(n)\})$. Here d is the distance between the two stationary end points, and OPT is the time needed by an optimal global strategy. Our strategy has the property that the robot which reaches its destination last always moves with maximum speed. Thus, the same bound as above also holds for the distance travelled.

1 Introduction

We envision a scenario in which two stationary devices (stations) with limited communication radii are placed within the plane. In order to provide communication between them, n mobile robots are deployed which form a chain capable of forwarding communication packets. As the stations, the robots have a limited communication range too. Assuming that the robots are arranged as an arbitrarily shaped, possibly winding chain in the beginning, the goal is to design and analyze a strategy for the mobile robots which minimizes the length of the chain. Each robot has to plan and perform its movement based on the positions of its chain neighbors solely, which are within a constant distance in the beginning— no global view, communication or long term memory is provided.

Unlike most strategies considered for similar problems, we want to use a continuous time model. Therefore, we are not given a classical round model, but rather all robots

* Partially supported by the EU within FP7-ICT-2007-1 under contract no. 215270 (FRONTS) and DFG-project "Smart Teams" within the SPP 1183 "Organic Computing" and International Graduate School Dynamic Intelligent Systems.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 168–182, 2010.

can perpetually and simultaneously measure and adjust their movement paths, leading to curves as trajectories for the robots. Although this model fits to real applications [1] and also has interesting and important theoretical aspects, surprisingly, to our knowledge, it has only once been considered theoretically for a formation problem [2]. The authors give an algorithm which gathers robots in one point in finite time, but they do not give any further runtime bounds. One reason for not using a continuous time model might be that completely different analysing techniques have to be applied, compared to usual discrete models. We are optimistic that the techniques for analysis which we develop in this paper have the potential to be applied to other continuous formation problems, e.g. the gathering problem.

We study a natural strategy, which we call MOVE-ON-BISECTOR: A robot moves in the direction of the bisector between its two neighbors at all times with maximum speed 1 (which equals the viewing range), until it reaches the straight line between them, and thenceforward stays on this straight line. We analyze this strategy and prove that it is valid in terms of connectedness: If neighbors are originally in distance at most 1, they will remain in distance at most 1 when performing the MOVE-ON-BISECTOR-strategy. Then we show bounds on the runtime needed until all robots are on the straight line between the stations. Since the robots move with velocity 1 as long as they have not reached the line between their neighbors, the finishing time is also the maximum distance a robot can travel.

In order to measure the quality of our strategy, we consider the time needed by our local strategy compared to the time needed by an optimal global strategy. The optimal global time is clearly lower bounded by the maximum distance between the initial robot positions and the final straight line. We will call this maximum distance *height* throughout the paper.

The problem at hand has been investigated before, but only in classical discrete models, which we describe below in the section on related work.

Our contribution. We initiate the study of a continuous time model for local robot formation problems by constructing a short communication chain. Our main result proves that the MOVE-ON-BISECTOR-strategy needs at most time $\mathcal{O}(min\{n, (OPT + d)\log(n)\})$, where OPT is the time needed by an optimal global strategy and d the distance between the two stations. This result implies an asymptotically optimal worst case time bound of $\mathcal{O}(n)$. In addition it shows that our continuous, local strategy is $\mathcal{O}(\log(n))$-competitive compared to a global optimal strategy, if d is sufficiently small.

Organization of the paper. We begin the next section by formally introducing the model and the MOVE-ON-BISECTOR-strategy. Section 3 is dedicated to the analysis of the strategy. The analysis is divided into three major parts: in Section 3.1 we show that the MOVE-ON-BISECTOR-strategy maintains a valid chain, in which the robots stay in distance 1 to their neighbors. Then we analyze the MOVE-ON-BISECTOR-strategy on input instances in which at least some robots are far away from their final destination. Clearly, also a global algorithm needs long to optimize those chains. In Section 3.3 we present this paper's main result. We show that input instances, which are solved fast by an optimal global algorithm, are also handled fast by MOVE-ON-BISECTOR. We conclude with some open questions.

Related work. The problem of building short communication chains locally has been considered before in discrete settings. In [3] an intuitive strategy has been presented: robots move synchronously to their neighbors' old mid position. In this strategy, the robots converge to the line between the stations in $\mathcal{O}(n^2 \log n)$ and $\Omega(n^2)$ steps. This result has been improved in [4] with a more sophisticated strategy that lets the robots come close to the line in only a linear number of steps in the worst case. Note that an optimal strategy might be a lot faster depending on the input instance and no bounds for this case are given. There is also some experimental work from the engineering point of view [1,5]. In [6] a local algorithm for a more general problem is considered: robots are distributed in the plane and have to shorten a communication network between several base stations. They have to base their decision on where to move in the next round on the relative position of all robots currently within distance 1.

Related problems are the gathering problem and the convergence problem, where robots are not required to form a line, but to gather in or to converge to some not predefined point in the plane. The focus is usually to limit the capabilities of the robots as far as possible and show that the task can be achieved in finite time, e.g. [7,8,9,10,11,12]. However, there are only few examples with provable runtime bounds such as [13], where an $\mathcal{O}(n^2)$ bound for a global algorithm is given. There are also some results for gathering [14] and convergence [15] with local view, but only one giving a runtime [16].

A strategy for the gathering problem which is similar to the MOVE-ON-BISECTOR-strategy has been considered in [2]. The authors also use the continuous time model, interpreting it as the limit of a discrete time model with the length of the time steps converging to 0. However, in contrast to our work it is only proven that it gathers the robots in finite time. None of the work above compares the runtime results of a specific instance to those of an optimal global algorithm on the same instance.

2 Problem Description

We consider a set of $n+2$ robots $v_0, v_1, \ldots, v_{n+1}$ in the two dimensional euclidean plane $\mathbb{R}^2$. The robots v_0 and v_{n+1} are stationary and will be referred to as *base stations* or simply *stations*, while we can control the movement of the remaining n robots $v_1, v_2, \ldots, v_n$. At the beginning, the robots form a chain, where each robot v_i is neighbor of the robots v_{i-1} and v_{i+1}. The chain may be arbitrarily winding in the beginning. The goal is to optimize the length of the communication chain in a distributed way. We are constrained in that the robots have a limited viewing range, which we set to 1. Therefore, the communication chain is *connected* if and only if, for each two neighbors in the chain, the distance between them is less than or equal to 1. We assume that the chain is connected at the beginning, and we say that a strategy for the robots is *valid* if it keeps the chain connected.

The robots solely use information from the current point of time (they are oblivious), share no common sense of direction and communicate by observing the positions of their two neighbors only. However, we require that the robots are capable of distinguishing their neighbors from the remaining robots in the communication chain (it is not necessary to distinguish the two neighbors from each other).

The continuous time model. In our continuous time model, time passes in a continuous way and is not modeled by discrete time steps. Thus, robots are able to continuously measure their neighbors' positions and adjust their trajectory and speed accordingly (keeping the speed limit of 1). In the MOVE-ON-BISECTOR-strategy, the direction in which a robot moves can change continuously. In contrast, the speed can also change rapidly in a non-continuous way. Both direction and speed depend only on the positions of the neighboring robots. We assume that a robot can measure these positions *without delay*: the direction in which a robot moves depends only on the positions of its neighbors at the same time.

In order to measure the quality of our algorithm, we determine the time until all robots have reached their final position. Assuming a maximum velocity of 1, this time is equal to the distance travelled by a robot which always moves with velocity 1. We will see that when using the MOVE-ON-BISECTOR-strategy, there is at least one robot for which this is true.

In our model we assume accuracy of the robots' actors and sensors in several aspects: measurements of the relative positions of a robot's neighbors (angle formed by a robot and its neighbours, distances to neighbors) are exact; adjustment of speed and direction is accurate; this adjustment is not delayed compared to the measurement. In the final Section 4 we will discuss our strategy in the light of inaccurate sensors and actors.

The MOVE-ON-BISECTOR-*Strategy.* In the MOVE-ON-BISECTOR strategy, robots that have not yet reached the straight line between their two neighbors will move along the bisector formed by the vectors pointing towards their two neighbors with maximum speed 1 (phase 1). Once they have reached this line, they adapt their velocity to stay on this (moving) line keeping the ratio between the distances to their neighbors constant (phase 2). Since the neighbors are also restricted to the maximum speed of 1, this is always possible: a robot will not have to move faster than with speed 1 to stay on this line and to keep the ratio. Note that one special situation may occur: if two neighboring robots v_i and v_j are at the same position at the same time, they both take the other's neighbor as their new neighbor, respectively. Then, both robots have the same neighbors and will stay together prospectively. For the sake of clarity, we will ignore such situations in our analysis.

Since robots which have reached the second phase stay in this phase until the end, all robots have reached the final line between the two stations as soon as all robots are in the second phase. Thus, the last robot reaching the second phase always moves with maximum speed.

Notions & Notation. Given a time $t \geq 0$, the position of robot v_i at this time is denoted by $v_i(t) \in \mathbb{R}^2$. If not stated otherwise, we will assume $v_0(0) = (0,0)$ and $v_{n+1}(0) = (d,0)$, $d \in \mathbb{R}_{\geq 0}$ denoting the distance between the two base stations. The vector connecting two neighboring robots v_{i-1} and v_i will be denoted by $w_i(t) := v_i(t) - v_{i-1}(t)$ for $i = 1,2,\ldots,n+1$. By $\alpha_i(t) \geq 0$ we denote the smaller of the two angles formed by the vectors $-w_i(t)$ and $w_{i+1}(t)$. We will furthermore denote the scalar product of two vectors a and b simply by $a \cdot b$ and the length of a vector a by $||a||$. A fixed placement of the robots (say their positions at a given time t) is called a *configuration*. Furthermore, we define two properties for a given configuration:

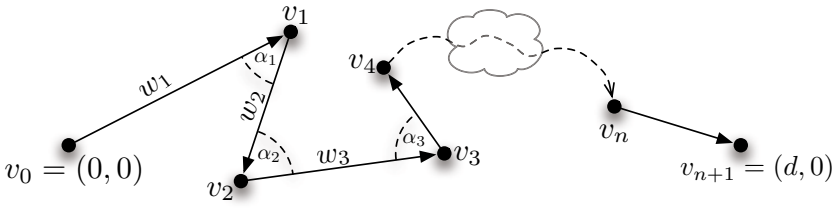


Fig. 1. Robots v_i positioned in the euclidean plane. For clarity we omitted the time parameter t.

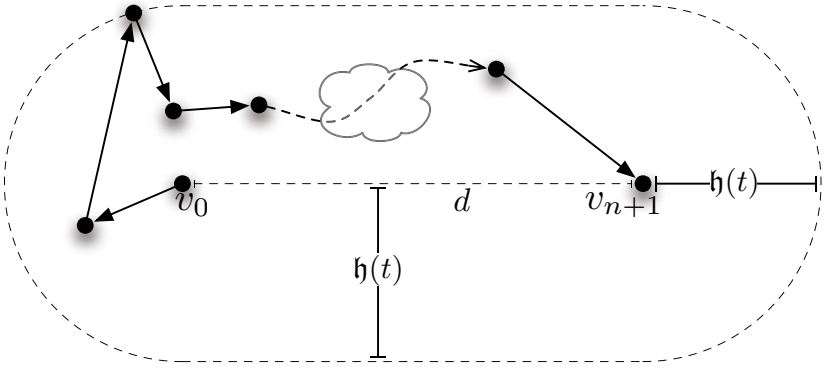


Fig. 2. Illustration of the height $\mathfrak{h}(t)$ of a configuration

Definition 1 (height). *The* height $\mathfrak{h}(t)$ *of a configuration at time t is the maximum distance between a robot in time t and the straight line between the stations.*

Definition 2 (length). *The* length $\mathfrak{l}(t)$ *of a configuration at a time t is defined as the sum of the distances between neighboring robots:* $\mathfrak{l}(t) := \sum_{i=1}^{n+1} ||w_i(t)||$.

Clearly, the height $h := \mathfrak{h}(0)$ is a lower bound for the time needed by an optimal global algorithm, since every algorithm needs to cover this distance. The length of a configuration is also a natural quantity to measure its quality: a winding chain is relatively long compared to a straight line. Since the distance between two robots may be at most 1, it holds that $h \leq \frac{l}{2}$ and $l \leq n+1$. See Figure 1 and Figure 2 for an illustration of the notions defined in this section.

3 Analysis

In this section, we will show that the MOVE-ON-BISECTOR-strategy is valid (Subsection 3.1) and analyze the time needed until all robots are positioned on the line between the two base stations (Subsections 3.2 and 3.3). We use the *length* and *height* of a given

configuration to show two upper bounds of $\mathcal{O}(l)$ and $\mathcal{O}((h+d)\log l)$, where d denotes the distance between the two base stations. The first bound is tight for configurations, in which an optimal global algorithm is slow. On other configurations this bound can be arbitrarily bad. Therefore, we show the second bound for this kind of input instances and combine the bounds to our main result of an upper bound of $\mathcal{O}(min\{n,(OPT+d)\log(n)\})$.

3.1 Validity of the Strategy

Let us first consider two robots v_i and v_j with $j > i$ at a time when neither v_i nor v_j have reached the line between their neighbors, but any robot v_k with $i < k < j$ has. That is, the robots v_k form a straight line between v_i and v_j. We will show that the distance between v_i and v_j decreases with non-negative speed. Given that all robots v_k between v_i and v_j maintain the ratio between the distances to their corresponding neighbors, this implies that the distance between *any* two neighboring robots is monotonically decreasing, and thus the chain stays connected and the MOVE-ON-BISECTOR-strategy is valid. We start by considering the case that both, v_i and v_j, are *mobile* (not base stations).

Lemma 1. *Given two* robots v_i *and* v_j *at an arbitrary time* t_0*, their distance decreases with speed* $\cos\frac{\alpha_i(t_0)}{2}+\cos\frac{\alpha_j(t_0)}{2} \geq 0$.

Proof. We define $D:\mathbb{R}_{\geq 0}\to\mathbb{R}^2, t\mapsto v_j-v_i$ and $\mathsf{d}:\mathbb{R}_{\geq 0}\to\mathbb{R}_{\geq 0}, t\mapsto ||D(t)||$. That is, $D(t)$ is the vector from v_i to v_j and $\mathsf{d}(t)$ the distance between v_i and v_j at time t. We want to show that $\mathsf{d}'(t_0) = -\left(\cos\frac{\alpha_i(t_0)}{2}+\cos\frac{\alpha_j(t_0)}{2}\right)$ for an arbitrary but fixed point of time t_0. We will refer to the x- and y-component of $D(t)\in\mathbb{R}^2$ in the following by $D_x(t)$ and $D_y(t)$ respectively.

By translating and rotating the coordinate system, we can w.l.o.g. assume $v_i(t_0) = (0,0)$ and $v_j(t_0) = (\mathsf{d}(t_0),0)$. Due to the definition of the MOVE-ON-BISECTOR strategy, the velocity vectors of v_i and v_j at time t_0 are given by:

$$v_i'(t_0) = \left(+\cos\frac{\alpha_i(t_0)}{2}, \pm\sin\frac{\alpha_i(t_0)}{2}\right)$$
$$v_j'(t_0) = \left(-\cos\frac{\alpha_j(t_0)}{2}, \pm\sin\frac{\alpha_j(t_0)}{2}\right)$$

See Figure 3 for an illustration.

Basic analysis now gives us the following equation for the first derivation of d at a time $t\in\mathbb{R}_{\geq 0}$[1]:

$$\mathsf{d}'(t) = \begin{pmatrix}\frac{D_x(t)}{\mathsf{d}(t)} & \frac{D_y(t)}{\mathsf{d}(t)}\end{pmatrix}\cdot\begin{pmatrix}D_x'(t)\\ D_y'(t)\end{pmatrix}$$

Due to the fact that $D_y(t_0) = 0$ and $D_x(t_0) = \mathsf{d}(t_0)$ we finally get

$$\begin{aligned}\mathsf{d}'(t_0) = D_x'(t_0) &= (v_j - v_i)'(t_0) = v_j'(t_0) - v_i'(t_0)\\ &= -\left(\cos\tfrac{\alpha_i(t_0)}{2}+\cos\tfrac{\alpha_j(t_0)}{2}\right)\end{aligned}$$

[1] Remember that we assume $\mathsf{d}(t)\neq 0$ (see the description of the MOVE-ON-BISECTOR-strategy in Subsection 2).

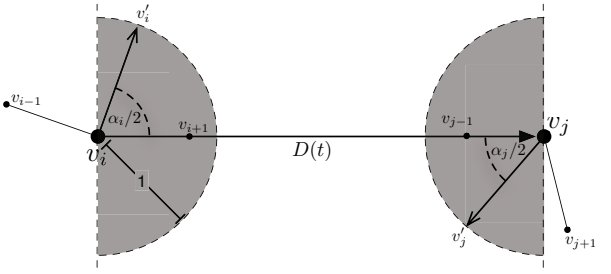


Fig. 3. Illustration of v_i's and v_j's velocity vectors v_i' and v_j'

Therefore, the distance between v_i and v_j changes at time t with speed $\cos(\frac{\alpha_i(t_0)}{2}) + \cos(\frac{\alpha_j(t_0)}{2})$. Furthermore, since we have $\alpha_i(t) \in [0,\pi]$ for any $t \in \mathbb{R}_{\geq 0}$ and $i \in \{1,\ldots,n\}$, this speed is indeed positive and the distance *decreases*. □

A similar result holds if either v_i or v_j is a base station. Since this can be proven completely analogously to Lemma 1, we will omit the proof and merely state the corresponding result.

Lemma 2. *Consider two robots v_i and v_j at an arbitrary time t_0, one of them being a base station and the other a robot not yet having reached the line between its neighbors. Then their distance decreases with speed* $\cos\frac{\alpha_j(t_0)}{2} \geq 0$. □

Now, we have the preliminaries to state the validity of the MOVE-ON-BISECTOR strategy.

Theorem 1. *The* MOVE-ON-BISECTOR *strategy is valid. That is, if the robot chain is connected at time t and all robots perform the* MOVE-ON-BISECTOR *strategy, the robot chain remains connected for any time $t' \geq t$.*

Proof. As described above, the statement follows immediately from Lemmas 1 and 2 conjoint with the fact that any robot, which has already reached the line between its neighbors, will move such that it maintains the ratio between the distances to its two neighbors. □

3.2 The $\mathcal{O}(l)$ Upper Bound

We continue by analyzing how long it will take for all robots to reach the straight line between the two stations. We will derive a time bound of $\mathcal{O}(l)$, l denoting the length of the robots' initial configuration. Because $h \leq l/2$ and $l = \mathcal{O}(n)$ (the distance of neighboring robots is bounded by 1), this immediately implies a linear bound $\mathcal{O}(n)$ on the time until the optimal configuration is reached. Since there are start configurations with a height of $\Omega(n)$, the MOVE-ON-BISECTOR-strategy is asymptotically optimal for worst case start configurations. The next section will show a tighter bound for configurations, where the height is relatively small compared to the length of the configuration. The result can then be compared to an optimal algorithm.

In the following, we will show that either the length l or height h of the robot chain decreases with constant speed. Since both are furthermore monotonically decreasing and bounded from below, this implies that the optimum configuration will be reached in time $\mathcal{O}(h+l) = \mathcal{O}(l)$. We begin with the monotonicity of the height.

Lemma 3. *The height of the robot chain is monotonically decreasing and bounded from below by 0.*

Proof. The lower bound is trivial, it follows directly from the definition of the robot chain's height. For the monotonicity, fix a time $t \in \mathbb{R}_{\geq 0}$ and consider the height $\mathfrak{h}(t)$ of the configuration at time t. Let B denote the line segment connecting both base stations and note that all robots are contained in the convex set $H := \{x \in \mathbb{R}^2 \mid dist(x,B) \leq \mathfrak{h}(t)\}$ of points having a distance of at most $\mathfrak{h}(t)$ to B. Let us consider an arbitrary robot v_k and its neighbors v_{k-1} and v_{k+1}. Since H is convex and all three robots lie in H, so does the bisector along which v_k moves. That is, v_k can not leave H. Since this argument applies to any robot, none of the robots can increase their distance to B beyond $\mathfrak{h}(t)$. This implies the monotonicity of the robot chain's height. □

Lemma 4. *The length of the robot chain decreases with speed* $2\sum_{i=1}^{n} \cos\frac{\alpha_i(t)}{2}$ *and is bounded from below by* d.

Proof. Since both base stations do not move, the length can obviously not fall below their distance d. Using the function $\mathfrak{l} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}, t \mapsto \mathfrak{l}(t)$ to refer to the chain's length at time t, it remains to show that $\mathfrak{l}'(t) = -2\sum_{i=1}^{n} \cos\frac{\alpha_i(t)}{2}$.

Fix a time $t \in \mathbb{R}_{\geq 0}$ and consider the robots $v_{i_1}, v_{i_2}, \ldots, v_{i_r}$ (for an $r \in \mathbb{N}$ and $i_s < i_{s+1} \forall s = 1, \ldots, r-1$) that have not yet reached the line between their neighbors. We make two observations:

- For any robot v_j of the remaining robots, it holds that $\alpha_j(t) = \pi$ and therefore $\cos\frac{\alpha_j(t)}{2} = 0$.
- Any of the remaining robots either lies on the line between some v_{i_s} and $v_{i_{s+1}}$ or on the line between one of the base stations and v_{i_1} or v_{i_r}. That is, setting $\mathfrak{l}_0(t) := ||v_0(t) - v_{i_1}(t)||$, $\mathfrak{l}_k(t) := ||v_{i_k}(t) - v_{i_{k+1}}(t)||$ $(k = 1, \ldots, r-1)$ and $\mathfrak{l}_r(t) := ||v_{i_r}(t) - v_{n+1}(t)||$, the length $\mathfrak{l}(t)$ of the chain is given by:

$$\mathfrak{l}(t) = \sum_{k=0}^{r} \mathfrak{l}_k(t)$$

Now, Lemmas 1 and 2 give us the derivations of these $\mathfrak{l}_k$, and therefore we have:

$$\begin{aligned}
\mathfrak{l}'(t) &= \mathfrak{l}_0'(t) + \sum_{k=1}^{r-1} \mathfrak{l}_k'(t) + \mathfrak{l}_r'(t) \\
&= -\cos\frac{\alpha_{i_1}(t)}{2} + \sum_{k=1}^{r-1} \left(-\cos\frac{\alpha_{i_k}(t)}{2} - \cos\frac{\alpha_{i_{k+1}}(t)}{2}\right) - \cos\frac{\alpha_{i_r}(t)}{2} \\
&= -2\sum_{k=1}^{r} \cos\frac{\alpha_{i_k}(t)}{2} = -2\sum_{i=1}^{n} \cos\frac{\alpha_i(t)}{2}.
\end{aligned}$$

□

Now we can prove an upper bound for the travelled distance in dependency of h and l, implying also a worst case upper bound.

Theorem 2. *When the* MOVE-ON-BISECTOR *strategy in the continuous model is performed, the maximum distance travelled by a robot is* $\frac{\sqrt{2}}{2}h + \sqrt{2}l$*, where* h *is the height and* l *the length of the robot chain in the start configuration.*

Proof. We will prove that in time $\frac{\sqrt{2}}{2}h + \sqrt{2}l$ all robots have reached their corresponding end positions. Given that the robots move with a maximum velocity of 1, this proves the theorem. To do so, we show that at any time either the height function $\mathfrak{h} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ or the length function $\mathfrak{l} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ are strictly decreasing *by a constant factor*. Together with Lemmas 3 and 4 (the monotonicity and non-negativity of $\mathfrak{l}$ and $\mathfrak{h}$) this proves the theorem.

So, let us consider an arbitrary time $t \in \mathbb{R}_{\geq 0}$. We distinguish two cases:

Case 1: $\exists i \in \{1, \ldots, n\} : \alpha_i(t) \leq \pi/2$

In this case, Lemma 4 states that:

$$\mathfrak{l}'(t) = -2 \sum_{k=1}^{n} \cos \frac{\alpha_k(t)}{2} \leq -2 \cos \frac{\alpha_i(t)}{2} \leq -2 \cos \frac{\pi}{4} = -\sqrt{2}$$

That is, the length of the robot chain decreases with a *constant* speed of at least $\sqrt{2}$.

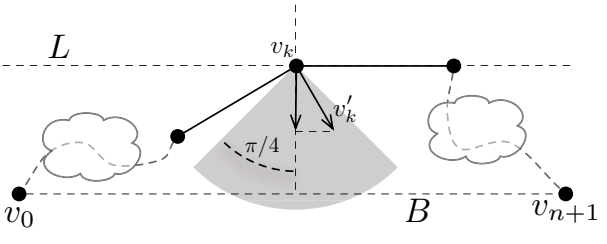


Fig. 4. If all angles α_i are larger than $\pi/2$, then the velocity vector of a "highest" robot v_k lies within the gray area. It therefore moves downwards with a speed of at least $\cos \pi/4$.

Case 2: $\forall i \in \{1, \ldots, n\} : \alpha_i(t) > \pi/2$

Using the terms from the proof of Lemma 3, consider a robot v_k with distance $\mathfrak{h}(t)$ to the line segment B connecting both base stations. Align the coordinate system such that the line L through $v_k(t)$ having distance $\mathfrak{h}(t)$ to B corresponds to the x axis and $v_k(t)$ to the origin. Figure 4 illustrates the situation.

We know that both neighbors of v_i must lie on the same side of L as B, w.l.o.g. let it be the lower side. Furthermore, because we have $\alpha_k(t) > \pi/2$, one neighbor must lie to the lower left and the other to the lower right of v_k. This implies that v_k's velocity vector is directed downwards, forming an angle of less than $\pi/4$ with the y-axis. Therefore, v_k moves with a speed of more than $\cos \frac{\pi}{4}$ downwards.

Since this holds for any extremal robot, we get $h'(t) < -\cos\frac{\pi}{4} = -\frac{1}{\sqrt{2}}$. That is, the height of the robot chain decreases with a constant speed of at least $\frac{\sqrt{2}}{2}$. □

Since $h \in \mathcal{O}(l)$, Theorem 2 gives an upper bound of $\mathcal{O}(l)$ for arbitrary start configurations. This result directly shows that the MOVE-ON-BISECTOR-strategy is asymptotically optimal for worst-case instances (Cor. 1), the measure which is usually used in the literature. Still, this bound can be arbitrarily worse than an optimal algorithm on specific instances. We will investigate these instances in the next section.

Corollary 1. *When the* MOVE-ON-BISECTOR*-strategy in the continuous model is performed, the maximum distance travelled by a robot is $\Theta(n)$ for a worst-case start configuration.*

Proof. Obviously it holds that $h \leq \frac{l}{2} \leq \frac{n+1}{2}$. For the lower bound, we can use a start configuration in which the stations share the position $(0,0)$ and $v_i(0) = v_{n+1-i}(0) = (0,i)$. Thus, the robot in the middle of the chain is in distance $\approx \frac{n}{2}$ of its end position and MOVE-ON-BISECTOR (as well as any global algorithm) needs at least this time until all robots have reached the line between the base stations. □

3.3 The $\mathcal{O}((h+d)\log l)$ Upper Bound

Assume we are given a configuration whose height is—relative to the length of the communication chain—very small. In this case, the upper bound of $\mathcal{O}(l)$ for our strategy can be arbitrarily larger than the time needed by an optimal strategy, which can be as small as h. But intuitively, given a long chain with a small height, the chain must be quite winding, yielding many relatively small angles α_i. The result is that the chain length does not only decrease at one robot, as we can only guarantee for arbitrary configurations, but there are many robots which reduce the length of the chain (Lemma 4).

For the proof of this upper bound, we will divide the chain into parts of length $\Theta(h+d)$ and show that each part must contain some curves. In particular, in each part, the sum of the angles $\alpha_i(t)$ must be by a constant smaller than in a straight line (Lemma 5). Lemma 6 transfers this result for each part to the sum of the angles of the whole chain. Having that the sum of the angles in the whole chain cannot be arbitrarily large, Lemma 7 yields the speed by which the length of the chain decreases. Since the number of parts is dependent on the length of the chain, the speed is also dependent on it. Theorem 3 finally gives the upper bound of $\mathcal{O}((d+h)\log l)$.

Lemma 5. *Let $\mathfrak{B}$ denote an arbitrary rectangular box containing the robots $v_{a-1}, v_a, v_{a+1}, \ldots, v_b$ (for $a,b \in \{1,\ldots,n+1\}$, $a<b$) at a given time $t \in \mathbb{R}_{>0}$ and let S be the diagonal length of the box. Then we have:*

$$\sum_{k=a}^{b} ||w_k(t)|| \geq \sqrt{2}\cdot S \Rightarrow \sum_{k=a}^{b-1} \alpha_k(t) \leq (b-a)\pi - \frac{\pi}{3}$$

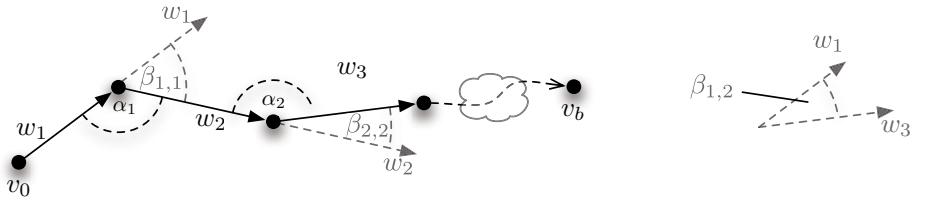


Fig. 5. Note that the angles $\beta_{i,j}$ are signed, e.g.: $\beta_{1,1} > 0$, $\beta_{2,2} < 0$, $\beta_{1,2} = \beta_{1,2} + \beta_{2,3} > 0$

Proof. For the sake of clarity, we will omit the time parameter t in the following. That is we write α_k, v_k and w_k instead of $\alpha_k(t)$, $v_k(t)$ and $w_k(t)$. Furthermore, we assume w.l.o.g. $a = 1$. Thus, we have to show $\sum_{k=1}^{b} ||w_k|| \geq \sqrt{2} \cdot S \Rightarrow \sum_{k=1}^{b-1} \alpha_k \leq (b-1)\pi - \frac{\pi}{3}$

Consider the function $\angle : \mathbb{R}^2 \times \mathbb{R}^2 \to]-\pi, \pi]$ that maps two vectors (w_i, w_j) to the signed angle of absolute value $\leq \pi$ formed by them (it is not important which direction is used as the positive angle, as long as it is equal for all pairs of vectors (w_i, w_j)). Note that we have $\alpha_k = \pi - |\angle(w_k, w_{k+1})|$ for all $k = 1, \ldots, b-1$. Let us define $\beta_{i,j} := \sum_{k=i}^{j} \angle(w_k, w_{k+1})$ and observe that $\angle(w_i, w_j) \equiv \beta_{i,j} \mod]-\pi, \pi]$. See Figure 5 for an illustration.

Let us now assume $\sum_{k=1}^{b} ||w_k(t)|| \geq \sqrt{2} \cdot S$ and consider the following two cases:

Case 1: $\exists i, j, 1 \leq i < j \leq b : |\beta_{i,j}| \geq \frac{\pi}{3}$

Intuitively, if the angle between two vectors in the chain is large, the sum of the inner angles α_k of the robots in between cannot be arbitrarily large. More formally,

$$\begin{aligned}
\sum_{k=1}^{b-1} \alpha_k &\leq (i-1)\pi + \sum_{k=i}^{j} \alpha_k + (b-1-j)\pi \\
&= (b+i-j-2)\pi + \sum_{k=i}^{j} (\pi - |\angle(w_k, w_{k+1})|) = (b-1)\pi - \sum_{k=i}^{j} |\angle(w_k, w_{k+1})| \\
&\leq (b-1)\pi - \left| \sum_{k=i}^{j} \angle(w_k, w_{k+1}) \right| = (b-1)\pi - |\beta_{i,j}| \leq (b-1)\pi - \frac{\pi}{3}
\end{aligned}$$

Thus, the lemma holds in this case.

Case 2: $\forall i, j, 1 \leq i < j \leq b : |\beta_{i,j}| < \frac{\pi}{3}$

We will show that this case cannot occur by showing that the vector connecting v_0 and v_b, which is equal to $\sum_{k=1}^{b} w_k$, would have to be longer than S, which is a contradiction to v_0 and v_b both lying in $\mathfrak{B}$.

We have $\angle(w_i, w_j) = \beta_{i,j}$ and $|\beta_{i,j}| < \frac{\pi}{3}$ for all $1 \leq i < j \leq b$. In the following, we will use that the squared length of a vector is equal to its scalar product with itself. Therefore:

$$\left|\left|\sum_{k=1}^{b} w_k\right|\right|^2 = \left(\sum_{k=1}^{b} w_k\right) \cdot \left(\sum_{k=1}^{b} w_k\right) = \sum_{1\le i,j\le b} w_i \cdot w_j = \sum_{1\le i,j\le b} ||w_i|| \cdot ||w_j|| \cdot \cos(\beta_{i,j})$$
$$> \sum_{1\le i,j\le b} ||w_i|| \cdot ||w_j|| \cdot \cos(\frac{\pi}{3}) = \cos(\frac{\pi}{3}) \sum_{1\le i,j\le b} ||w_i|| \cdot ||w_j||$$
$$= \frac{1}{2}\left(\sum_{k=1}^{b} ||w_k||\right)^2 \ge \frac{1}{2} \cdot (\sqrt{2}S)^2 = S^2$$

This implies $||\sum_{k=1}^{b} w_k|| > S$, leading to the desired contradiction. □

Dividing the chain in parts of length at least $\sqrt{2}$ times the diagonal of the height box, Lemma 5 shows that each of the parts must contain some "small" angles. The robots at these angles therefore shorten the length of the chain. The following lemma shows that using the technique of dividing the chain into parts yields an upper bound on the sum of the angles α_i of the chain.

Lemma 6. *Let S denote the diagonal length of the robots' height-box at a given time t. Then we have:*

$$\sum_{k=1}^{n} \alpha_k(t) \le n\pi - \frac{\pi}{3}\left\lfloor \frac{\mathfrak{l}(t)}{2\sqrt{2}S} \right\rfloor$$

Proof. As in the proof for Lemma 5, we will omit the time parameter t in the following.

First note that we have $||w_k|| \le S$, because all robots lie inside the height-box. This allows us to recursively define indices $1 = a_0 < a_1 < \ldots < a_m \le n+1$ by demanding $a_i \in \mathbb{N}$ to be minimal with $\sum_{k=a_{i-1}}^{a_i} ||w_k|| \in [\sqrt{2}S, (\sqrt{2}+1)S[$. That is, we divide the chain at time t in m parts, where $v_{a_{i-1}}$ and v_{a_i} bound part i. v_{a_i} is the first robot in the chain such that the length of part i is at least $\sqrt{2}S$. Furthermore, since $||w_{a_i}|| \le S$, the length of part i is at most $\sqrt{2}S + S \le 2\sqrt{2}S$, which implies $m \ge \left\lfloor \frac{\mathfrak{l}}{2\sqrt{2}S} \right\rfloor$. Since we have $\sum_{k=a_{i-1}}^{a_i} ||w_k|| \ge \sqrt{2}S$ for all $i = 1, \ldots, m$, by Lemma 5 we get $\sum_{k=a_{i-1}}^{a_i-1} \alpha_k \le (a_i - a_{i-1})\pi - \frac{\pi}{3}$. We now compute:

$$\sum_{k=1}^{a_m-1} \alpha_k = \sum_{i=1}^{m} \sum_{k=a_{i-1}}^{a_i-1} \alpha_k \le \sum_{i=1}^{m} \left((a_i - a_{i-1})\pi - \frac{\pi}{3}\right)$$
$$= \pi \sum_{i=1}^{m} (a_i - a_{i-1}) - \frac{\pi}{3}m = (a_m - a_0)\pi - \frac{\pi}{3}m$$
$$= (a_m - 1)\pi - \frac{\pi}{3}m$$

This implies $\sum_{k=1}^{n} \alpha_k \le n\pi - \frac{\pi}{3}m \le n\pi - \frac{\pi}{3}\left\lfloor \frac{\mathfrak{l}}{2\sqrt{2}S} \right\rfloor$, as the lemma states. □

Using that the sum of the angles α_i is bounded, we can now give a lower bound for the speed by which the chain length decreases, which is linear in the current number of parts and therefore the length of the chain. Instead of the current number of parts,

which cannot be determined exactly only knowing the length of the chain, we use a lower bound for the number of parts.

Lemma 7. *The length of the robot chain decreases at least with speed* $\frac{2}{3}\left\lfloor \frac{\mathfrak{l}(t)}{2\sqrt{2}S} \right\rfloor$.

Proof. Fix a time $t \in \mathbb{R}_{\geq 0}$. By Lemma 4, the chain length decreases with a speed of $2\sum_{k=1}^{n} \cos\frac{\alpha_k(t)}{2}$. Using that $\cos(x)$ is lower bounded by $1-\frac{2}{\pi}x$ for all $x \in [0,\pi/2]$ and by Lemma 6 we get:

$$\begin{aligned}\mathfrak{l}'(t) &= -2\sum_{k=1}^{n} \cos\frac{\alpha_k(t)}{2} \leq -2\sum_{k=1}^{n}\left(1-\frac{\alpha_k(t)}{\pi}\right) = -2n+\frac{2}{\pi}\sum_{k=1}^{n}\alpha_k(t)\\ &\leq -2n+\frac{2}{\pi}\left(n\pi-\frac{\pi}{3}\left\lfloor\frac{\mathfrak{l}(t)}{2\sqrt{2}S}\right\rfloor\right) = -\frac{2}{3}\left\lfloor\frac{\mathfrak{l}(t)}{2\sqrt{2}S}\right\rfloor. \qquad \square\end{aligned}$$

Now we can finally state our main result.

Theorem 3. *When the* Move-On-Bisector *strategy in the continuous model is performed, the maximum distance travelled by a robot is* $\mathcal{O}((h+d)\log(l))$*, where* h *is the height and* l *the length of the robot chain in the start configuration.*

Proof. Set $m^* := \left\lfloor \frac{l}{2\sqrt{2}S} \right\rfloor$ and let us define m^* time-phases $\mathfrak{p}_i := [t_{i-1}, t_i]$ for $i = 1\ldots, m^*$ by setting $t_0 := 0$ and t_i for $i > 0$ to the time when we have $\mathfrak{l}(t_i) = (m^* - i + 1)\cdot 2\sqrt{2}S$. That is, during one phase $\mathfrak{p}_i$, the chain length is reduced by exactly $2\sqrt{2}S$ and thus in phase i, the chain must consist of at least m^* parts as defined in Lemma 6. Note that these t_i are well-defined, because by Lemma 7, in phase $\mathfrak{p}_i$ the chain length decreases with a speed of at least $\frac{2}{3}\left\lfloor\frac{\mathfrak{l}(t_i)}{2\sqrt{2}S}\right\rfloor = \frac{2}{3}\cdot(m^*-i+1)$ (which is a constant for fixed i). Furthermore, Lemma 7 gives us an upper bound on the length of each single phase $\mathfrak{p}_i$:

$$t_i - t_{i-1} \leq \frac{\mathfrak{l}(t_{i-1}) - \mathfrak{l}(t_i)}{\frac{2}{3}(m^*-i+1)} \leq \frac{2\sqrt{2}S}{\frac{2}{3}(m^*-i+1)}$$

This allows us to give an upper bound to the time when the last phase ends:

$$\begin{aligned}t_{m^*} &= \sum_{i=1}^{m^*}(t_i - t_{i-1}) \leq 3\sqrt{2}S\sum_{i=1}^{m^*}\frac{1}{m^*-i+1}\\ &= 3\sqrt{2}S\sum_{i=1}^{m^*} i^{-1} < 3\sqrt{2}S\cdot(\ln m^* + 1)\end{aligned}$$

Now consider the situation after time $t \geq t_{m^*}$. We have $\mathfrak{l}(t_{m^*}) = (m^* - m^* + 1)2\sqrt{2}S = 2\sqrt{2}S$. By Theorem 2, from now on it takes time at most $\frac{\sqrt{2}}{2}\mathfrak{h}(t_{m^*}) + \sqrt{2}\mathfrak{l}(t_{m^*}) \leq \frac{\sqrt{2}}{2}h + 4S$ for the robots to reach the optimal configuration. Together with the bound on t_{m^*} and with $S = \mathcal{O}(h+d)$, this yields a maximum time (and therefore travelled distance) of until the optimal configuration is reached.

$$
\begin{aligned}
&3\sqrt{2}\cdot S\cdot(\ln m^*+1)+\frac{\sqrt{2}}{2}h+4S\\
\leq&3\sqrt{2}\cdot S\cdot\left(\ln\left(\frac{l}{2\sqrt{2}S}\right)+1\right)+\frac{\sqrt{2}}{2}h+4S\\
=&3\sqrt{2}\cdot S\cdot(\ln l-\ln(2\sqrt{2}S)+1)+\frac{\sqrt{2}}{2}h+4S\\
=&\mathcal{O}(S\cdot\ln l)+\frac{\sqrt{2}}{2}h+4S=\mathcal{O}((h+d)\ln l)
\end{aligned}
$$

□

Corollary 2. MOVE-ON-BISECTOR *runs in time* $\mathcal{O}(min\{n,(OPT+d)\log n\})$. □

A consequence of this result is that for $d\in\mathcal{O}(h)$ our local algorithm is by at most a logarithmic factor slower than an optimal global algorithm.

4 Outlook

We initiated the study of the continuous time model for the robot chain problem. Furthermore we introduced the idea to compare a local algorithm to an optimal global algorithm for the same instance.

We showed the runtime of our algorithm to be $\mathcal{O}(min\{n,(OPT+d)\log(n)\})$. Future work includes improving our upper bounds for the algorithm as well as lower bounds for optimal global algorithms. Furthermore, we want to apply the developed techniques for the continuous time model and the concept of comparing local algorithms to optimal global ones to further formation problems, such as gathering and convergence to a point in the plane as well as more complex tasks like building short two-dimensional communication infrastructures like trees.

It is an interesting problem to investigate to which extend our strategy is robust under inaccuracies of sensors and actors as described in the chapter on the continuous time model in Section 1. How to formally model such inaccuracies? For which input instances is our strategy robust? Do we have to modify the strategy? We certainly have to assume that input instances have the property that neighboring robots have distance at most $1-\gamma$, where $\gamma\in(0,1)$ is chosen dependent on parameters describing the accuracy.

References

1. Nguyen, H., Farrington, N., Pezeshkian, N., Gupta, A., Spector, J.M.: Autonomous communication relays for tactical robots. In: Proc. of the 11th International Conference on Advanced Robotics (ICAR), pp. 35–40 (2003)
2. Gordon, N., Wagner, I.A., Bruckstein, A.M.: Gathering multiple robotic a(ge)nts with limited sensing capabilities. In: Ant Colony, Optimization and Swarm Intelligence, pp. 142–153 (2004)
3. Dynia, M., Kutyłowski, J., Lorek, P., Meyer auf der Heide, F.: Maintaining Communication Between an Explorer and a Base Station. IFIP International Federation for Information Processing, vol. 216, pp. 137–146. Springer, Boston (2006)

4. Kutyłowski, J., Meyer auf der Heide, F.: Optimal strategies for maintaining a chain of relays between an explorer and a base camp. Theoretical Computer Science 410(36), 3391–3405 (2009)
5. Nguyen, H.G., Pezeshkian, N., Gupta, A., Farrington, N.: Maintaining communication link for a robot operating in a hazardous environment. In: Proc. of the 10th Int. Conf. on Robotics and Remote Systems for Hazardous Environments, American Nuclear Society (2004)
6. Meyer auf der Heide, F., Schneider, B.: Local strategies for connecting stations by small robotic networks. In: IFIP International Federation for Information Processing, Biologically-Inspired Collaborative Computing, September 2008, vol. 268, pp. 95–104. Springer, Boston (2008)
7. Mataric, M.: Designing emergent behaviors: From local interactions to collective intelligence. In: Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats, vol. 2, pp. 432–441 (1992)
8. Dieudonné, Y., Petit, F.: Self-stabilizing deterministic gathering. In: Algorithmic Aspects of Wireless Sensor Networks, pp. 230–241 (2009)
9. Souissi, S., Défago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Principles of Distributed Systems, pp. 333–349 (2006)
10. Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: Gathering autonomous mobile robots with dynamic compasses: An optimal result. In: Distributed Computing, pp. 298–312 (2007)
11. Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots. In: SODA 2004: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics, pp. 1070–1078 (2004)
12. Czyzowicz, J., Gasieniec, L., Pelc, A.: Gathering few fat mobile robots in the plane. Theoretical Computer Science, Principles of Distributed Systems 410(6-7), 481–499 (2009)
13. Cohen, R., Peleg, D.: Convergence properties of the gravitational algorithm in asynchronous robot systems. SIAM Journal on Computing 34(6), 1516–1528 (2005)
14. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Transactions on Robotics and Automation 15(5), 818–828 (1999)
15. Ando, H., Suzuki, Y., Yamashita, M.: Formation agreement problems for synchronous mobile robotswith limited visibility. In: Proc. IEEE Syp. of Intelligent Control, pp. 453–460 (1995)
16. Degener, B., Kempkes, B., Meyer auf der Heide, F.: A local $O(n^2)$ gathering algorithm. In: Symposium on Parallelism in Algorithms and Architectures (2010)

Optimal Deterministic Ring Exploration with Oblivious Asynchronous Robots

Anissa Lamani, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil

Université Pierre et Marie Curie - Paris 6, LIP6-CNRS 7606, France

Abstract. We consider the problem of exploring an anonymous unoriented ring of size n by k identical, oblivious, asynchronous mobile robots, that are unable to communicate, yet have the ability to sense their environment and take decisions based on their local view. Previous works in this weak scenario prove that k must not divide n for a deterministic solution to exist. Also, it is known that the minimum number of robots (either deterministic or probabilistic) to explore a ring of size n is 4. An upper bound of 17 robots holds in the deterministic case while 4 probabilistic robots are sufficient. In this paper, we close the complexity gap in the deterministic setting, by proving that no deterministic exploration is feasible with less than five robots, and that five robots are sufficient for any n that is coprime with five. Our protocol completes exploration in $O(n)$ robot moves, which is also optimal.

Keywords: Robots, Anonymity, Obliviousness, Exploration, Asynchronous system, Ring.

1 Introduction

Recent research focused on systems of autonomous mobile entities (that are hereafter referred to as *robots*) that have to collaborate in order to accomplish collective tasks. Two universes have been studied: the continuous euclidean space [8,14,4] where the robots entities can freely move on a plane, and the discrete universe in which space is partitioned into a finite number of locations, conventionally represented by a graph, where the nodes represent the possible locations that a robot can take and the edges the possibility for a robot to move from one location to the other [7,11,2,1,10,9,5,6,3]. In this paper we pursue research in the discrete universe and focus on the exploration problem when the network is an anonymous unoriented ring, using a team of autonomous mobile robots. The robots we consider are unable to communicate, however they can sense their environment and take decisions according to their local view. We assume anonymous and uniform robots (*i.e* they execute the same protocol and there is no way to distinguish between them using their appearance). In addition they are oblivious, *i.e* they do not remember their past actions. In this context, robots asynchronously operate in cycles of three phases: look, compute and move phases. In the first phase, robots observe their environment in order to get the position of all the other robots in the ring. In the second phase, they perform a

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 183–196, 2010.

local computation using the previously obtained view and decide on their target destination to which they will move in the last phase.

Related Work. In the discrete model, two main problems are investigated assuming very weak asynchronous, identical, and oblivious robots: the gathering and the exploration problem. In the gathering problem, robots have to gather in one location not known in advance *i.e* there exists an instant $t > 0$ where all robots share the same location (one node of the ring). In the exploration problem, robots have to explore a given graph, every node of the graph must be visited by at least one robot and the protocol eventually terminates (that is, all robots are idle).

For the problem of gathering in the discrete robot model, the aforementioned weak assumptions have been introduced in [10]. The authors proved that the gathering problem is not feasible in some symmetric configurations and proposed a protocol based on breaking the symmetry of the system. By contrast in [9], the authors proposed a gathering protocol that exploits this symmetry for a large number of robots ($k > 18$) closing the open problem of characterizing symmetric situations on the ring which admit a gathering.

For the exploration problem, the fact that the robots have to stop after the exploration process implies that the robots somehow have to remember which part of the graph has been explored. Nevertheless, in this weak scenario, robots have no memory and thus are unable to remember the various steps taken before. In addition, they are unable to communicate explicitly, therefore the positions of the other robots remain the only way to distinguish different stages of the exploration process. The main complexity measure here is the minimal number of robots necessary in order to explore a given graph. It is clear that a single robot is not sufficient for the exploration in the case where it is not allowed to use labels. In [6], it has been shown that $\Omega(n)$ robots are necessary in order to explore trees of size n, however, when the maximum degree of the tree is equal to three then the exploration can be done with a sub-linear robot complexity. In the case where the graph is a ring, it has been shown in [5] that k (the number of robots) must not divide n (the size of the ring) to enable a deterministic solution. This implies that for a general n, $\log(n)$ robots are necessary. The authors also present in [5] a deterministic protocol using 17 robots for every n that is coprime with 17. By contrast, [3] presents a probabilistic exploration algorithm for a ring topology of size $n > 8$. Four probabilistic robots are proved optimal since the same paper shows that no protocol (probabilistic or deterministic) can explore a ring with three robots.

Contribution. In this paper, we close the complexity gap in the deterministic setting. In more details, we prove that there exists no deterministic protocol that can explore an even sized ring with $k \leq 4$ robots. This impossibility result is written for the ATOM model [14] where robots execute their look, compute and move phases in an atomic manner, and thus extend naturally in the non-atomic CORDA model. We complement the result with a deterministic protocol using five robots and performing in the fully asynchronous non-atomic CORDA model

[13] (provided that five and n are coprime). The total number of robot moves is upper bounded by $O(n)$, which is trivially optimal.

2 Model and Preliminaries

We consider a distributed system of mobile robots scattered on a ring of n nodes u_0,u_1,..., $u_{(n-1)}$ such as u_i is connected to both $u_{(i-1)}$ and $u_{(i+1)}$. The ring is assumed to be anonymous *i.e* there is no way to distinguish the nodes or the edges (i.e. there is no available labeling). In addition, the ring is unoriented *i.e* given two neighbors, it is impossible to determine which node is on the right or on the left of the other. On this ring k robots collaborate to explore all the nodes of the ring. The robots are identical *i.e* they cannot be distinguished using their appearance and all of them execute the same protocol. Additionally, the robots are oblivious *i.e* they have no memory of their past actions. We assume the robots do not communicate in a explicit way. However, they have the ability to sense their environment and see the position of the other robots. Each robot can detect whether several robots are on the same node or not, this ability is called *multiplicity detection*. Robots operate in three phase cycles: Look, Compute and Move. During the Look phase robots take a snapshot of their environment. The collected information (position of the other robots) are used in the compute phase in which robots decide to move or to stay idle. In the last phase (move phase) they may move to one of their adjacent nodes towards the target destination computed in the previous phase.

At some time t, a subset of robots are activated by an abstract entity called *scheduler*. The scheduler can be seen as an external entity which selects some robots for the execution. In the following we assume that the scheduler is fair *i.e* each robot is activated infinitely many times. Two computational models exist: The *ATOM model* [14], in which synchronous cycles are executed in atomic way *i.e* the robots selected by the scheduler at the beginning of a cycle execute synchronously the full cycle, and the *CORDA model* [13] in which the scheduler is allowed to interleave different phases (For instance one robot can perform a look operation while another is moving). The model considered in our case is the CORDA model with the following constraint: the Move operation is instantaneous *i.e* when a robot takes a snapshot of its environment, it sees the other robots on nodes and not on edges. Nevertheless, since the scheduler is allowed to interleave the operations, a robot can move according to an outdated view (during the computation phase, some robots have moved).

In the following we assume that initially every node of the ring contains at most one robot. During the system execution a subset of robots are activated and move to other nodes. During the Look phase, the activated robots take a snapshot of their environment in order to see the position of the other robots. For a robot located at node u_i at time t, the snapshot result also called the view at node u_i at time t, is defined by the two following sequences: $C^{+i}(t) =< d_i(t)d_{i+1}(t)...d_{i+n-1}(t) >$ and $C^{-i}(t) =< d_i(t)d_{i-1}(t)...d_{i-(n-1)}(t) >$ where $d_j(t)$ denotes the multiplicity of robots on the node u_j at instant t taking

an arbitrarily orientation of the ring. $d_j \geq 1, \forall j \in [1, n]$ if and only if u_j is occupied by at least one robot, $d_j = 0$ otherwise. When $d_j(t) = 0$, the node u_j is said to be empty at instant t, when $d_j(t) = 1$, we say that the node u_j is occupied at instant t, otherwise we say that there is a *tower* on u_j at instant t.

The view at u_i is said to be *symmetric* at instant t if and only if $C^{+i}(t) = C^{-i}(t)$. Otherwise, the view of u_i is said to be *asymmetric*. When the view at a node u_i is symmetric, both edges incident to the node u_i look identical to the robot located at that node. In this case we assume the worst scenario allowing the adversary to take the decision on the direction to be taken.

A configuration of the system can be described as the union of the views at each node in the system. In order to define symmetric configurations we borrow some definitions of [5]. Let $\Delta^+(t) = \{C^{+i}(t), \forall i = 1, n\}$ and let $\Delta^-(t) = \{C^{-i}(t), \forall i = 1, n\}$. As advocated in [5] each one of the sets $\Delta^+(t)$ and $\Delta^-(t)$ contains a maximal sequence. A configuration is said *s*ymmetric if the maximal sequences in $\Delta^+(t)$ and $\Delta^-(t)$ are equal, and asymmetric otherwise.

Problem to Be Solved. The problem considered here is the exploration problem, where k robots have to collaborate and explore a ring of size n before stopping forever. A protocol P solves the exploration problem if and only if the following conditions are satisfied:

1. **Safety.** Every node is visited by at least one robot.
2. **Termination.** The algorithm eventually stops.

3 Impossibility Result

It has been shown [5] that no deterministic team of k robots can explore a ring of size n when k divides n. Also, it is known [3] that no exploration protocol (deterministic or probabilistic) is possible when $0 \leq k < 4$. We now observe that when a single robot is activated at a time, a trivial deterministic variation of the protocol of [3] (that uses four robots and randomization to break symmetry in some situations) matches the lower bound. So, our leveraging of the lower bound result of [3] to four robots considers the case when several robots can be activated at the same time.

Lemma 1. *There exists no deterministic protocol in the ATOM (and CORDA) model for exploring a ring of an even size (n) with four robots.*

Proof: The proof is by contradiction. We assume that there exists a deterministic protocol with four robots that can explore a ring and terminate. Then, we start from an admissible initial configuration where no two robots are located on the same node and derive executions that never satisfy the exploration specification.

We consider the two similar configurations shown in figures 1 and 2. As the configurations contain two axes of symmetry, the four robots $R1$, $R2$, $R3$ and $R4$ have identical views, which means that if they are activated simultaneously, they will exhibit the same behavior.

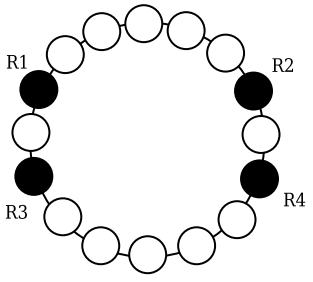


Fig. 1. Instance of configuration

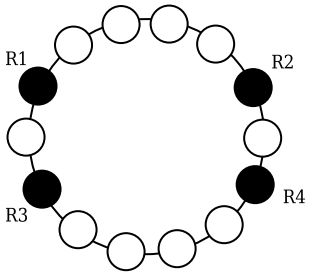


Fig. 2. Instance of configuration

1. *Suppose that every robot move towards its adjacent node in the same direction of their neighbor robot at distance* 2. Assume that all the robots are activated at the same time by the scheduler, then two towers are created (one with $R1$ and $R3$, the other with $R2$ and $R4$). From this point onwards, we assume that $R1$ and $R3$ are always activated simultaneously (and likewise for $R2$ and $R4$). As a result, the fours robots now behave as two robots. As it was shown in [3], no team of two robots can explore the ring, and thus the initial protocol does not perform a ring exploration either.
2. *Suppose that every robot move towards its adjacent node in the opposite direction of their neighbor robot at distance* 2. If the robots move back to their position, then the protocol can never stop since the robots can go back and forth indefinitely. In the case where the robots keep moving away then two cases are possible :
   - *The number of nodes between* $R1$ *and* $R2$ *is even* (the same for $R3$ and $R4$ see figure 2) in this case, by moving away from the robots that are at an odd distance from them, the configuration reached is similar to the one shown in figure 3 in which $R1$ and $R2$ are neighbors (the same for $R3$ and $R4$). Since the robots cannot go back (the protocol may never stop), the only move that they can perform is moving towards their neighbor: $R1$ moves towards $R2$ and vice versa (the same for $R3$ and $R4$), however, in the case where the four robots are activated at the same time, the two robots that are neighbors simply exchange their positions, and the configuration remains unchanged. As a result, no progress is made towards completion of the exploration task.
   - *The number of nodes between* $R1$ *and* $R2$ *is odd* (the same for $R3$ and $R4$, see figure 1). In this case k divides n, and [5] proved that the exploration problem in impossible to solve in this setting.

From the cases above, we can deduct that no deterministic exploration is possible using four robots when the size of the ring is even. When the size of the ring is odd, the following Lemma show that no protocol with four robots can explore the ring in the CORDA model. Due to lack of space, the proof is presented in the appendix.

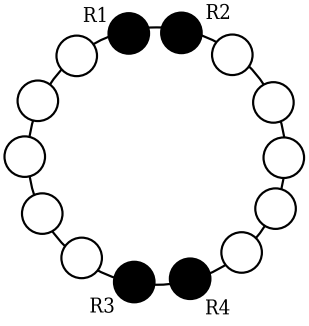


Fig. 3. Instance of configuration

4 Ring Exploration in CORDA Model

In this section we propose the ring exploration in Corda model with only five robots provided the size of the ring (n) and the number of robots are co-prime. Before detailing our algorithm we introduce some definitions.

A *hole* is the maximal set of consecutive empty nodes. The *size of a hole* is the number of nodes that compose it, the *border of the hole* are the two empty nodes who are part of this hole, having one robot as a neighbor.

An **inter-distance** d is the minimum distance taken among distances between each pair of distinct robots (in term of the number of edges). Given a configuration of interdistance d, a **d.block** is any maximal elementary path in which there is a robot every d edges. The *border of a d.block* are the two extremal robots of the d.block. The *size of a d.block* is the number of robots in the d.block. A robot not in a d.block is *isolated*.

Given a configuration of interdistance 1, a *tower-chain* consists of a 1.block of size 3 followed by an empty node followed by a tower.

Our protocol consists of three distinct phases orchestrated as shown in Algorithm 1:

- **Block Module.** The aim of this phase is to drag all the robots into one single 1.block starting from any initial configuration that does not contain a tower.
- **Tower Module.** Starting from a configuration that contains a single 1.block, one tower is created such that an elected robot will benefit from an orientation of the ring allowing to explore the ring in the last phase.
- **Tower-chain Module.** In this phase, starting from a configuration with a single tower, one robot is elected in order to explore the ring.

Note that once a configuration with a tower-chain is reached, the ring has been explored and the protocol terminates. Remark also that robots are able to distinguish between the phase since each phase has different particularities. In the first phase all the configurations are tower-less and do not contain 1.block of

Algorithm 1. The orchestration of the algorithm

```
1: if the five robots do not form a tower-chain then
2:     if the configuration contains neither a tower nor a single 1.block then
3:         Execute Block Module
4:     else
5:         if the configuration contains a single 1.block then
6:             Execute Tower Module
7:         else
8:             Execute Tower-chain Module
9:         end if
10:    end if
11: end if
```

size 5. In the second phase, configurations contain a single 1.block of size 5. And finally, in the last phase, the configurations contain a single tower.

The following section details and analyzes the complexity of the previous modules.

Block Module Description and Analysis. The aim of this phase is to reach a configuration with no tower where there is a single 1.block that contains all the five robots. This phase is described in Algorithm 2.

Lemma 2. *If the configuration at instant t contains neither a single 1.block nor a tower, then the configuration at instant $t+1$ is tower-less.*

Proof: We prove in this section that, if a robot moves, it moves always to an empty node to avoid the creation of towers. We suppose that the configuration at instant t is C. The configuration C doesn't contain any tower and satisfies one of the following cases:

- C contains at least one isolated robot: Two cases are possible according to the number of d.blocks:

  1. There is a single d.block: in this case the isolated robots that are the closest to the d.block are allowed to move, they move to an empty node (see line 4). As it is an isolated robot, there are at least d empty nodes between it and the target d.block. In another hand, since $d \geq 1$, by moving, no tower is created at instant $t+1$.
  2. There are two d.blocks: in this case, the configuration contains a single isolated robot (there are five robots on the ring), this robot is the only one allowed to move (see line 9, 10), when it moves, it does to an empty node toward one of the two blocks depending of the symmetry of the configuration (there is no other robot between it and the target d.block and there are at least d empty nodes between it and the d.blocks – otherwise it would be part of them). Thus, no tower is created at instant $t+1$.

Algorithm 2. Procedure: Block Module executed by robot r

```
1: if the configuration contains at least one isolated robot then
2:     if the configuration contains a single d.block then
3:         if r is isolated robot and r is closest neighbor to the d.block then
4:             Move toward the d.block taking the shortest hole
5:         end if
6:     else
7:         if the configuration contains two d.block then
8:             if the configuration is symmetric then
9:                 if r is isolated then
10:                    Move toward one of the two d.blocks
11:                end if
12:            else
13:                if r is isolated then
14:                    Move toward the closest d.block
15:                end if
16:            end if
17:        end if
18:    end if
19: else
20:    if the configuration contains a single d.block and d > 1 then
21:        if r is at the border of the d.block then
22:            Move toward the adjacent node in the direction of the d.block
23:        end if
24:    else
25:        if the configuration contains two d.blocks then
26:            if r is in the smallest d.block and r is the closest to the biggest d.block
                   then
27:                Move toward the biggest d.block
28:            end if
29:        end if
30:    end if
31: end if
```

– C contains no isolated robots in the configuration: two cases are possible according to the number of d.blocks:

1. C contains a single d.block: in this case the robots at the border of this d.block are the only robots allowed to move, if they do, they move to an empty node toward the d.block they belong to. This guarantee is given by the condition $d > 1$ (see line 18). Hence, no tower is created at instant $t + 1$.
2. C contains two d.blocks: in this case the two d.blocks have different sizes (since there are no isolated robots and the number of robots is odd). Robots in the smallest d.block and closest to the biggest d.block move toward the target d.block taking the hole that separate them from one extremity of the biggest d.block. Since the size of the hole is at least

equal to the inter-distance (otherwise robots are in the same d.block), no tower is created at instant $t+1$.

Overall no tower is created at instant $t+1$.

Lemma 3. *Starting from a configuration with no tower the system reaches a configuration with a single* 1.*block after* $O(n)$ *move operations.*

Proof: Two cases are possible according to the type of the starting configuration denoted in the following C:

1. C contains at least one isolated robot: in this case, the robots allowed to move are always the isolated ones, and their destination is the closest d.block $(line3, 4)$, or one of the two d.blocks in the case of symmetry $(line9, 10)$. Hence three cases are possible according to the number of isolated robots:

   – The configuration C contains a single isolated robot. This robot is the only one allowed to move and its destination is the d.block. After its move the distance between it and the target d.block decreases. Since this robot remains the only isolated robot in the configuration (robots in the d.blocks do not move when there is at least one isolated robot), it is the only one that keeps moving to the same target d.block. Therefore, after a finite time, the robot joins the d.block.

   In order to compute the maximum number of moves (NBM) we consider the the worst case: the number of nodes between the two robots at the border of the d.block is odd. Consider the Figure 4.

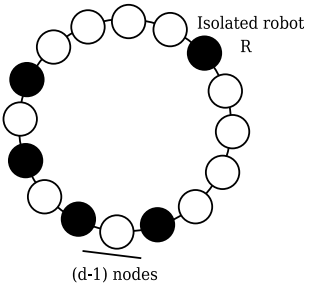


Fig. 4. Instance of configuration

Let's compute the number of nodes between the d.block and the isolated robot. In order to do this, we will substract from the size of the ring, the occupied nodes and the empty nodes between the robots in the d.block.

The obtained result gives the sum of the empty nodes between the isolated robot and the d.block at each side. Thus, we have to divide it by two to obtain the distance between the isolated robot and the d.block.

Note that the isolated robot is going to join the d.block hence it will advance until it reaches the same distance as the other robots in the d.block. Thus, in order to calculate the number of moves of the isolated robot, we have to substract from $n - [k + 3 * (d - 1)]$, the number of nodes between any two robots in the d.block.

The NBM is hence given by the following formula:

$$NBM = [\frac{n - [k + 3 * (d - 1)]}{2}] - (d - 1) \quad (1)$$

– The configuration contains two isolated robots: in this case, at least one of these two isolated robots is allowed to move. If there is a single robot that is the closest to the d.block, then this robot is the only one that moves, its destination is the single d.block (see line 3, 4). At each move, the robot becomes even closer to the d.block, hence after a finite time, it reaches the d.block and the configuration contains a single isolated robot. In the other case (there are two robots allowed to move, let them be $R1$ and $R2$), whatever the choice of the scheduler in interleaving the different operations, at least one of these two isolated robots moves towards the d.block and hence becoming even closer. Note that if robots move at the same time both reduce their distance to the d.block.

Suppose the worst case: a single robot moves. Let $R1$ be this robot. Two cases are possible: If the robot that does not move, $R2$, has an up to date view of the configuration, then $R1$ becomes the closest robot to the d.block and hence it is the only one allowed to further move. From this point onward the proof is similar to the case 1. If $R2$ has an absolete view then $R2$ may also move. Consequently, either it is at the same distance as $R1$ from the d.block or $R1$ is the closest one to the d.block. The proof goes on recursively until at least one of the two robots reaches the d.block.

Note that the worst case happens when the two robots are at the maximum distance from the d.block and the number of empty nodes between these two robots is minimal (equal to d otherwise they form a d.block). It follows that the maximum number of moves robots perform in this case is given by the following formula:

$$NBM = n - (k + 4 * (d - 1) + d) \quad (2)$$

– The configuration contains three isolated robots. From the above cases above it follows that after a finite time all isolated robots join the d.block. The maximum number of moves in this case is performed when the three robots are at a maximum distance from the d.block and the distance between them is minimal and it is equal to d. This number is given by the following formula:

$$NBM = n - (k + 3 * (d - 1) + 2d) + [n - (k + 3 * (d - 1))]/2 - (d - 1) \quad (3)$$

Overall the number of isolated robots decreases until the configuration contains only d.blocks.

2. The configuration contains only d.blocks: Two sub-cases are possible according to the number of d.blocks:
   - The configuration contains two d.blocks. In this case the two d.blocks have different size and the smallest d.block moves towards the biggest one ($line 24, 25$). Consequently, whatever the choice of the scheduler at least one of the two robots moves towards the biggest one, when it does the configuration changes and contains a single d.block with isolated robots. However, it has been shown in case 1 that in this case and after a finite time all the isolated robots join the d.block. Hence the number of d.blocks decreases and the configuration contains a single d.block. The maximum number of moves is defined by the following formula and it happens when the small block is at a maximum distance from the biggest d.block:
   $$NBM = n - (k + 5 * (d - 1)) \tag{4}$$
   - The configuration contains a single d.block: if $d > 1$ then there is at least one single node between each robot, and in this case, depending on the choice of the scheduler at least one of the two robots that are at the border of the d.block moves to its adjacent node in the direction of the d.block it belongs to ($line 19, 20$). Thus, the inter-distance decreases and the configuration reached contains isolated robots. However, once the configuration with a single d.block is reached, the maximum number of moves that are performed in order to reach another configuration with a single $(d-1)$.block $\forall d$ such as $d > 1$ is constant and is equal to 7. If one of the two robots at the border of the d.block moves, then one $(d-1)$.block is created and all the other robots move towards it starting with the closest one, since all the robots were at the same distance, the closest one performs one move to reach the $(d-1)$.block, the next robots performs two moves and the third one (the last one) performs three moves to reach the d.block. Hence, if we sum up all these moves taking in account the first move of the robot that creates the $(d-1)$.block, then the total number of displacements is the following: $1+(1+...+k-2)$ and is equal to 7. In the case where the two robots at the border of the d.block move at the same time, two $(d-1)$.block are created. The isolated robot that is on this axes of symmetry chooses one of them by moving towards it, when it moves it joins the chosen $(d-1)$.block and the configuration contains two $(d-1)$.blocks. However, in this case, only one robot is allowed to move (the closest one (see line $24, 25$), since the two robots in the smallest $(d-1)$.block are at different distance from the biggest one). This robot performs two moves to join the biggest $(d-1)$.block, the same for the second robot. Thus if we sum up the moves that were performed $(2 + 1 + 2 + 2)$, the total number is equal to 7. Since $d > 1$ the same process repeats the system reaches a configuration with a single 1.block. In this case the number of moves is given by the following formula:
   $$NBM = (d - 1) * 7 \tag{5}$$

Since $d < n/5 - 1$ the total number of moves in order to reach a 1.*block* configuration starting from any tower-less configuration is $O(n)$.

Tower Module Description and Analysis. This phase begins when the configuration contains a single 1.block. It aims at creating a tower in order to give a virtual orientation to the ring such as the elected robot accomplish the exploration task in the last phase. This phase is described in Algorithm 3.

Algorithm 3. Procedure Tower Module executed by robot r

```
1: if r is on the axes of symmetry then
2:     Move toward one of my neighbors
3: end if
```

Lemma 4. *Let C be the configuration that contains a single 1.block of size* 5*. If C is the configuration at instant t, then the configuration at instant $t+1$ contains a single tower.*

Tower-Chain Module Description and Analysis. In this phase, one robot is elected in order to explore the ring. The exploration begins when the configuration contains a tower and is done when a tower-chain is created. This phase is described in Algorithm 4.

Algorithm 4. Procedure tower-chain Module executed by robot r

```
1: if the configuration doesn't contain a chain-tower then
2:     if r between the tower and the 1.block then
3:         Move toward my adjacent node in the opposite direction of the tower
4:     end if
5: end if
```

Lemma 5. *Starting from a configuration with a single tower, the system reaches a configuration that contains a chain-tower after $O(n)$ move operations and all the nodes have been explored.*

5 Conclusion

In this paper, we focused on the exploration problem in an undirected ring. We proved that no deterministic protocol can explore such a graph using k robots such as $k \leq 4$ if the ring is of even size. On the other hand, we provided a non-atomic completely asynchronous algorithm that uses only five robots for completing exploration provided that n and k are coprime. Our solution is thus optimal with respect to the number of robots. As exploration requires $O(n)$ robots moves, it is also optimal in time. We would like to mention two interesting open questions raised by our work:

1. The impossibility result of [5] shows that k must not divide n (for arbitrary values of k and n), while our impossibility result shows that k must be coprime with n (for a specific value of k: 4). We conjecture that the impossibility result of [5] can be extended to any k and n that are not coprime and such that $n > k$.
2. The summary of results presented below sheds new light on the respective powers of the ATOM and CORDA models on the one hand, and on the power of random choice on the other hand. Our impossibility result holds for even sized rings even in the ATOM model, so going probabilistic [3] is the only way to pass the five robots barrier. By contrast, when the size is odd, probabilities do not help (the companion technical report [12] shows that a determinsitic ATOM protocol can exist with four robots). Further investigating the relationships between the sheduling models (CORDA/ATOM) and the protocol power (probabilistic/deterministic) is an intriguing path for future research.

Reference	k and n	Model	Deterministic	Nb of robots
[5]	$n \neq ck$	CORDA	Yes	$k \geq 17$
[3]	$n > 8$	ATOM	No	$k = 4$
This paper	$n \neq ck$	CORDA	Yes	$k = 5$
[12]	n,k are coprime n is odd	ATOM	Yes	$k = 4$

Acknowledgements

This work is supported by ANR projects SHAMAN, ALADDIN, and R-DISCOVER.

References

1. Das, S., Flocchini, P., Kutten, S., Nayak, A., Santoro, N.: Map construction of unknown graphs by multiple agents. Theoretical Computer Science 385, 34–48 (2007)
2. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. Theoretical Computer Science 355, 315–326 (2006)
3. Devismes, S., Petit, F., Tixeuil, S.: Optimal probabilistic ring exploration by asynchronous oblivious robots. In: Proceedings of SIROCCO 2009 (2009)
4. Dieudonné, Y., Labbani-Igbida, O., Petit, F.: Circle formation of weak mobile robots. TAAS 3(16) (2008)
5. Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Computing without communicating: Ring exploration by asynchronous oblivious robots. In: Tovar, E., Tsigas, P., Fouchal, H. (eds.) OPODIS 2007. LNCS, vol. 4878, pp. 105–118. Springer, Heidelberg (2007)
6. Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Remembering without memory: Tree exploration by asynchronous oblivious robots. In: Shvartsman, A.A., Felber, P. (eds.) SIROCCO 2008. LNCS, vol. 5058, pp. 33–47. Springer, Heidelberg (2008)

7. Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Multiple mobile agent rendezvous in a ring. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 599–608. Springer, Heidelberg (2004)
8. Flocchini, P., Prencipe, P., Santoro, N., Widmayer, P.: Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. Theoretical Computer Science 407, 412–447 (2008)
9. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: Gathering of asynchronous oblivious robots on a ring. In: OPODIS, pp. 446–462 (2008)
10. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theoretical Computer Science 390, 27–39 (2008)
11. Kowalski, D., Pelc, A.: Polynomial deterministic rendezvous in arbitrary graphs. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 644–656. Springer, Heidelberg (2004)
12. Lamani, A., Potop-Butucaru, M., Tixeuil, S.: Optimal deterministic ring exploration with oblivious asynchronous robots. CoRR abs/0910.0832 (2009)
13. Prencipe, G.: CORDA: Distributed coordination of a set of autonomous mobile robots. In: Proc. 4th European Research Seminar on Advances in Distributed Systems (ERSADS 2001), Bertinoro, Italy, May 2001, pp. 185–190 (2001)
14. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal on Computing 28(4), 1347–1363 (1999)

Maximum Interference of Random Sensors on a Line

Evangelos Kranakis[1], Danny Krizanc[2], Lata Narayanan[3], and Ladislav Stacho[4]

[1] School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada
[2] Department of Mathematics and Computer Science, Wesleyan University, Middletown CT 06459, USA
[3] Department of Computer Science and Software Engineering, Concordia University, Montreal, QC, H3G 1M8, Canada
[4] Department of Mathematics, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada

Abstract. Consider n sensors whose positions are represented by n uniform, independent and identically distributed random variables assuming values in the open unit interval $(0, 1)$. A natural way to guarantee connectivity in the resulting sensor network is to assign to each sensor as range the maximum of the two possible distances to its two neighbors. The interference at a given sensor is defined as the number of sensors that have this sensor within their range. In this paper we prove that the expected maximum interference is $\Omega(\ln \ln n)$, and that for any $\epsilon > 0$, it is $O((\ln n)^{1/2+\epsilon})$.

1 Introduction

The broadcast nature of wireless communication implies that interference with other transmissions is inevitable. Interference can be caused by sources inside or outside the system and comes in many forms. Co-channel interference is caused by other wireless devices transmitting on the same frequency channel. Such interference can make it impossible for a receiver to decode a transmission unless the signal power of the intended source is significantly higher than the combined strength of the signal received from the interfering sensors. Wireless devices are designed to admit a certain maximum level of interference. It is therefore crucial to understand the maximum possible interference in a wireless network.

In this paper we study the expected maximum interference for n sensors placed at random in the *highway model*. According to this model, n sensors are represented by n uniform, independent and identically distributed random variables in the open unit interval $(0, 1)$. Using standard notation (see [2]), let the corresponding order statistics be $X_{1:n} < X_{2:n} < \cdots < X_{n:n}$. The position, x_i, of the ith sensor is determined by event $X_{i:n} = x_i$ which is the value of the ith order statistic $X_{i:n}$.

Since only nodes whose transmissions can reach a node can cause interference at it, an important way to manage interference is by the use of topology control algorithms. In particular, one can assign transmission ranges to nodes with the

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 197–210, 2010.

objective of minimizing interference. On the other hand, the assignment of transmission ranges should also ensure that the network is connected. In the highway model, a natural algorithm is to assign as transmission range to a sensor the maximum distance between its two immediate (from left and right) neighbors since this is the minimum range required to attain connectivity.

Definition 1. *We assign as range of the ith sensor, denoted by* r_i*, the maximum of the two possible distances with its two neighbors, namely the random variable defined by* $\max\{X_{i:n}-X_{i-1:n}, X_{i+1:n}-X_{i:n}\}$*, where we use the notation* $X_{0:n}=0$ *and* $X_{n+1:n}=1$*.*

We are interested in studying the resulting *interference* among the n sensors. Intuitively, the interference for each sensor i is defined as the number of sensors that have the sensor i within their range. Formally, we consider the following sets of sensors.

Definition 2. *For each* i*, define the sets* $S^+(i), S^-(i)$ *of sensors* j *to the right (i.e.,* $j > i$*) and left (i.e.,* $j < i$*) of* x_i*, respectively, which have the sensor* i *within their range. Further, let* $S(i) = S^+(i) \cup S^-(i)$*.*

The interference is now defined as follows.

Definition 3. *Define the random variables* $I^+(i) := |S^+(i)|$*,* $I^-(i) := |S^-(i)|$ *and* $I(i) := I^+(i) + I^-(i)$*. For each* i*,* $I(i)$ *is the interference at sensor* i*.*

We are also interested in the random variable that specifies the neighborhood size (or proximity), the number of sensors that are in sensor j's transmission range.

Definition 4. *For each sensor* j *define the sets* $\Gamma^+(j), \Gamma^-(j)$ *of sensors* i *to the right (i.e.,* $i > j$*) and left (i.e.,* $i < j$*) which are (downstream) neighbors of* j*. Let* $\Gamma(j) = \Gamma^+(j) \cup \Gamma^-(j)$ *be the set of neighbors of* j*.*

Observe that

$$\Gamma(j) = \begin{cases} \Gamma^+(j) \cup \{j-1\} \text{ if } X_{j+1:n} - X_{j:n} < X_{j:n} - X_{j-1:n} \\ \Gamma^-(j) \cup \{j+1\} \text{ if } X_{j+1:n} - X_{j:n} > X_{j:n} - X_{j-1:n}. \end{cases}$$

Definition 5. *Define the random variables* $N^+(j) := |\Gamma^+(j)|, N^-(j) := |\Gamma^-(j)|$ *which is the number of sensors* i *which are within* j*'s range to the right and left of* i*, respectively. Let* $N(j) = N^+(j) + N^-(j)$*. For each* j*, we call* $N(j)$ *the neighborhood size of* j*.*

Let I be the random variable equal to $\max_i I(i)$. We are interesting in obtaining bounds on $E(I)$, the expected maximum interference experienced by any of the sensors.

1.1 Related Work

Several papers study interference and network performance degradation. [7] considers the throughput of wireless networks under two models of interference: one

is a protocol model that assumes interference to be an all-or-nothing phenomenon and the other a physical model that considers the impact of interfering transmissions on the signal-to-noise ratio. Motivated by this, [9] defined the concept of *conflict graph* (a graph indicating which groups of nodes interfere and hence cannot be active at the same time) and study what is the maximum throughput that can be supported by a wireless network given a specific placement of wireless nodes in physical space and a specific traffic workload.

[5] proposes connectivity preserving and spanner constructions which are interference optimal. [11] consider the *average interference* problem while maintaining connectivity and give a sharp $O(\ln n)$ upper and lower bound. Closely related to our study is the following problem first proposed by [10].

> Given n nodes in the plane. Connect the nodes by a spanning tree. For each node v we construct a disk centering at v with radius equal to the distance to v's furthest neighbor in the spanning tree. The interference of a node v is then defined as the number of disks that include node v (not counting the disk of v itself). Find a spanning tree that minimizes the maximum interference.

Choosing transmission radii which minimize the maximum interference while maintaining a connected symmetric communication graph is shown by [4] to be NP-complete. In addition, [8] gives an algorithm which yields a maximum interference in $O(\sqrt{n})$. An open problem that remains is to narrow the gap between this upper bound and the lower bound. For the case of points on a line [14] show that if nodes are distributed as an exponential node chain, the algorithm described in Section 1 has maximum interference $\Omega(n)$. They proceed to give an $n^{1/4}$-approximation algorithm for the problem.

[3] show that for broadcasting (one-to-all), gossiping (all-to-all), and symmetric gossiping (symmetric all-to-all) the problem of minimizing the maximum interference experienced by any node in the network is hard to approximate within better than a logarithmic factor, unless NP admits slightly superpolynomial time algorithms. They also show that any approximation algorithm for the problem of minimizing the total transmission power assigned to the nodes in order to guarantee any of the above communication patterns, can be transformed, by maintaining the same performance ratio, into an approximation algorithm for the problem of minimizing the total interference experienced by all the nodes in the network.

In addition, it is worth mentioning that the distribution of random sensors in a unit interval has been investigated in the past by the probability theory community. For example, using the main result of [6] (see also [1], [12] and [13]) concerning the extremes of the inter-arrival times of the Poisson process we see that

$$\lim_{n\to\infty} \Pr\left[\max_i\{X_{i+1:n} - X_{i:n}\} \le \frac{x + \ln n}{n}\right] = \exp[-e^{-x}],$$

for all x. Therefore we have that $\lim_{n\to\infty} \Pr\left[\max_i\{X_{i+1:n} - X_{i:n}\} \ge \frac{\ln n}{n}\right] = 1 - 1/e$, which implies that $E[\max_i X_{i+1:n} - X_{i:n}] \in \Omega(\ln n/n)$. Although this

result provides information about the expected size of the maximum gap between adjacent sensors it does not immediately imply anything about the expected value of the maximum interference.

1.2 Results of the Paper

We study a model where sensors are represented by n uniform, independent and identically random variables in the open unit interval $(0, 1)$. We assign to each sensor as range the maximum of the two possible distances with its two neighbors (including endpoints $0, 1$). In section 2 we show a number of useful lemmas and preliminary results including showing that the expected interference of a given sensor is in $O(1)$ (see Theorem 1) and giving a bound on the probability the maximum interference exceeds $\ln n$ (see Theorem 2). In section 3 we prove upper and lower bounds on $E(I)$. Theorem 3 shows that the expected maximum interference is $\Omega(\ln \ln n)$, while in Theorem 4 we show that for any $\epsilon > 0$ the expected maximum interference is $O((\ln n)^{1/2+\epsilon})$. This is in contrast to the result of [14] that the maximum interference for n sensors distributed on a line and connected in the same manner is $\Omega(n)$ in the worst case.

2 Preliminary Results

In this section we prove a number of lemmas that will be useful in proving our main results as well as some preliminary bounds on the expected interference and on the probability of the maximum interference exceeding $\ln n$. We begin our study by analyzing the relationship between interference and neighborhood size.

Lemma 1

$$\sum_j E[N(j)] = \sum_i E[I(i)]. \tag{1}$$

Proof. For each pair i, j of sensors consider the indicator random variable $X_{i,j}$ which indicates with 1 that sensor i is within j's range. Now observe that the following identities are valid.

$$E\left[\sum_j N(j)\right] = E\left[\sum_j |\{i : \ i \text{ is within } j\text{'s range}\}|\right] = E\left[\sum_{i,j} X_{i,j}\right]$$
$$= E\left[\sum_i |\{j : \ i \text{ is within } j\text{'s range}\}|\right] = E\left[\sum_i I(i)\right].$$

This proves the lemma.

Another useful fact is encapsulated in the following lemma.

Lemma 2. *Consider a constant $c > 0$. With probability at least $1 - 1/n^c$, any interval of length $\geq \frac{c \ln n}{n}$ contains a sensor.*

Proof. Consider an interval Δ of length $\frac{c \ln n}{n}$. Then we have that

$$\begin{aligned}\Pr[\Delta \text{ contains no sensor}] &= \Pr[\forall i (X_i \notin \Delta)] \\ &= (\Pr[X_i \notin \Delta])^n \\ &= \left(1 - \frac{c \ln n}{n}\right)^n \\ &\leq \frac{1}{n^c},\end{aligned}$$

which proves the lemma.

The assignment of ranges as above enforces a natural bound on the interference which we clarify in the next lemma.

Lemma 3. *For any sensor i we have*

1. *if $I^+(i) \geq k$ then there is a sensor $j > i+1$ such that*
$$2^k(x_{i+1} - x_i) \leq x_{j+1} - x_j,$$
2. *if $I^-(i) \geq k$ then there is a sensor $j < i-1$ such that*
$$2^k(x_i - x_{i-1}) \leq x_j - x_{j-1}.$$

Proof. First we prove Part 1. Assume that $I^+(i) = k$. Suppose that $j \in S^+(i)$. Since $i+1 < j$ and i is within j's range we have that $x_j - x_i \leq x_{j+1} - x_j$. This is easily seen to be equivalent to the inequality

$$2(x_j - x_i) \leq x_{j+1} - x_i.$$

Next suppose that we have k sensors $j_1, j_2, \ldots, j_k \in S^+(i)$ such that $j_1 < j_2 < \cdots < j_k$. Using the previous inequality for each of these k positions we see that

$$\begin{aligned} 2(x_{j_1} - x_i) &\leq x_{j_1+1} - x_i \\ 2(x_{j_2} - x_i) &\leq x_{j_2+1} - x_i \\ &\vdots \quad\quad \vdots \quad\quad \vdots \\ 2(x_{j_r} - x_i) &\leq x_{j_r+1} - x_i \\ &\vdots \quad\quad \vdots \quad\quad \vdots \\ 2(x_{j_k} - x_i) &\leq x_{j_k+1} - x_i. \end{aligned} \tag{2}$$

Using Inequalities 2 and induction we see easily that

$$\begin{aligned} x_{j_k+1} - x_{j_k} &\geq x_{j_k} - x_i \\ &\geq x_{j_{k-1}+1} - x_i \\ &\geq 2(x_{j_{k-1}} - x_i) \text{ (from Inequality 2)} \\ &\geq 2(x_{j_{k-2}+1} - x_i) \\ &\geq 2^2(x_{j_{k-2}} - x_i) \text{ (from Inequality 2)} \\ &\vdots \\ &\geq 2^k(x_{i+1} - x_i), \end{aligned}$$

which proves Part 1. The proof of Part 2 is similar.

Definition 6. *We call* range *the random variable* $R_{i,j:n} := X_{j:n} - X_{i:n}$*, which is defined for* $i < j$.

The probability distribution of the range (see [2][Formula 2.5.21, page 33]) is given by the identity below

$$\Pr[X_{j:n} - X_{i:n} = w] = \frac{n!}{(j-i-1)!(n-j+i)!} w^{j-i-1}(1-w)^{n-j+i}. \quad (3)$$

Using this and elementary calculations we see that

$$\Pr[X_{j:n} - X_{i:n} > r] = \int_r^1 \frac{n!}{(j-i-1)!(n-j+i)!} w^{j-i-1}(1-w)^{n-j+i} dw.$$

Elementary calculations using integration by parts yields the following identity for the range of the order statistics

$$\Pr[X_{i+k:n} - X_{i:n} > r] = \sum_{s=0}^{k-1} \binom{n}{s} r^s (1-r)^{n-s}, \quad (4)$$

for any k and i such that $i + k \leq n$.

Lemma 4. *We have the following identities for* $k \geq 1$

$$\Pr[X_{i:n} - X_{i-1:n} > X_{i+k:n} - X_{i:n}] = \frac{\binom{n}{k}}{\binom{2n}{k}} \quad \textit{(if } i + k \leq n\textit{)}$$

$$\Pr[X_{i+1:n} - X_{i:n} > X_{i:n} - X_{i-k:n}] = \frac{\binom{n}{k}}{\binom{2n}{k}} \quad \textit{(if } i - k \geq 1\textit{)}$$

Proof. We prove only the first identity. The second is proved in exactly the same manner. Using Identity 3 from order statistics (see [2]) with $j = i + k$ we derive

$$\begin{aligned}
&\Pr[X_{i:n} - X_{i-1:n} > X_{i+k:n} - X_{i:n}] \\
&= \int_0^1 \Pr[X_{i:n} - X_{i-1:n} > r \mid X_{i+k:n} - X_{i:n} = r] \Pr[X_{i+k:n} - X_{i:n} = r] dr \\
&= \int_0^1 (1-r)^n \frac{n!}{(k-1)!(n-k)!} r^{k-1}(1-r)^{n-k} dr \\
&= k\binom{n}{k} \int_0^1 (1-r)^{2n-k} r^{k-1} dr.
\end{aligned}$$

The last integral can be computed easily using the following recursive identity which is derived easily using integration by parts

$$\int_0^1 (1-r)^{2n-k} r^{k-1} dr = \frac{k-1}{2n-k+1} \int_0^1 (1-r)^{2n-k+1} r^{k-2} dr$$

which yields

$$\int_0^1 (1-r)^{2n-k} r^{k-1} dr = \frac{1}{k\binom{2n}{k}}.$$

Substituting this in the previous integral proves the lemma.

Another idea that will prove useful is to look at the ratio $\frac{R_{i,i+1:n}}{R_{j,j+1:n}}$ of ranges.

Lemma 5. *For all $i < j \leq n$ we have that*

$$\Pr\left[\frac{R_{i,i+1:n}}{R_{j,j+1:n}} > k\right] \leq \frac{1}{k}.$$

Proof. Observe that

$$\begin{aligned}
&\Pr\left[\frac{R_{i,i+1:n}}{R_{j,j+1:n}} > k\right] \\
&= \Pr[R_{i,i+1:n} > kR_{j,j+1:n}] \\
&= \int_0^{1/k} \Pr[R_{i,i+1:n} > kR_{j,j+1:n} \mid R_{j,j+1:n} = r] \Pr[R_{j,j+1:n} = r] dr \\
&= n \int_0^{1/k} \Pr[R_{i,i+1:n} > kR_{j,j+1:n} \mid R_{j,j+1:n} = r](1-r)^{n-1} dr \\
&= n \int_0^{1/k} \Pr[R_{i,i+1:n} > kr](1-r)^{n-1} dr \\
&= n \int_0^{1/k} (1-kr)^n (1-r)^{n-1} dr.
\end{aligned}$$

Now observe that

$$\int_0^{1/k} (1-kr)^n (1-r)^{n-1} dr \leq \int_0^{1/k} (1-kr)^n dr = \frac{1}{k(n+1)},$$

which proves the lemma.

We are also in a position to prove the following more precise result concerning the interference of random sensors.

Lemma 6. *For $c > 2$, for all i and k,* $\Pr[I(i) > k] \leq \max\{2c \ln n/2^k, 1/n^{c-2}\}$.

Proof. Without loss of generality we consider only $I^+(i)$. We prove that for any constant $c > 2$ we have that

$$\Pr[(\exists i \leq n)(\exists j > i + c \ln n) j \in S^+(i)] \leq \frac{1}{n^{c-2}}.$$

Indeed, for $j > i + 1$ we know that

$$\Pr[j \in S^+(i)] = \frac{\binom{n}{j-i}}{\binom{2n}{j-i}}.$$

Consider the event $E:\ (\exists i \leq n)(\exists j > i + c \ln n) j \in S^+(i)$. We have that

$$\begin{aligned}
\Pr[(\exists i \leq n)(\exists j > i + c \ln n) j \in S^+(i)] &\leq \sum_i \sum_{j > i + c \ln n} \Pr[j \in S^+(i)] \\
&\leq \sum_i \sum_{j > i + c \ln n} \frac{\binom{n}{j-i}}{\binom{2n}{j-i}} \\
&\leq \frac{1}{n^{c-2}}.
\end{aligned}$$

Note that

$$\Pr[I^+(i) > k] = \Pr[I^+(i) > k \mid \overline{E}]\Pr[\overline{E}] + \Pr[I^+(i) > k \mid E]\Pr[E].$$

If $k \geq c \ln n$ we have $\Pr[I^+(i) > k \mid \overline{E}] = 0$ and it follows that $\Pr[I^+(i) > k] \leq \Pr[E] \leq 1/n^{c-2}$. For $k < c \ln n$ we can use the main argument of Lemma 3 to derive

$$\begin{aligned}\Pr[I^+(i) > k \mid \overline{E}] &\leq \Pr\left[\exists j > i+1 \left(2^k \leq \frac{X_{j+1:n} - X_{j:n}}{X_{i+1:n} - X_{i:n}}\right) \mid \overline{E}\right] \\ &\leq \sum_{i+1<j<i+c\ln n} \Pr\left[2^k \leq \frac{X_{j+1:n} - X_{j:n}}{X_{i+1:n} - X_{i:n}}\right] \\ &\leq \frac{c \ln n}{2^k},\end{aligned}$$

where the last inequality follows from Lemma 5. The lemma now follows.

2.1 Expected Interference of a Given Sensor

We are now ready to show that the expected interference of a given sensor is a constant independent of n.

Theorem 1. *$E[I(i)] \in O(1)$, for all i.*

Proof. First we prove a lemma that analyzes the expected neighborhood size of a given sensor.

Lemma 7. *For all j, $E[N(j)] \in O(1)$.*

Proof. Observe that the event $N^+(j) > k$ is equivalent to the event "j has more than k neighbors to its right" and similarly for the event $N^-(j) > k$. Therefore $N(j) > k$ if and only if j has at least $k+1$ neighbors. Therefore j either has at least $\lceil k/2 \rceil$ neighbors either to its left or to its right. It follows from Lemma 4 that

$$\begin{aligned}\Pr[N(j) > k] &= \Pr[N^+(j) > \lceil k/2 \rceil] + \Pr[N^-(j) > \lceil k/2 \rceil] \\ &\leq 2\frac{\binom{n}{\lceil k/2 \rceil}}{\binom{2n}{\lceil k/2 \rceil}}.\end{aligned}$$

In particular,

$$\begin{aligned}E[N(j)] &= \sum_{k \geq 1} \Pr[N(j) > k] \\ &\leq 2\sum_{k} \frac{\binom{n}{\lceil k/2 \rceil}}{\binom{2n}{\lceil k/2 \rceil}} \\ &\leq \sum_{k} \frac{1}{2^{\lceil k/2 \rceil}},\end{aligned}$$

which proves the lemma.

To prove the main theorem we now argue as follows. Join the endpoints of the unit interval to form a circle with perimeter of length 1 where the sensors lie on the perimeter of the circle. Consider the corresponding random variables $N_C^+(i), N_C^-(i), N_C(i), I_C^+(i), I_C^-(i), I_C(i)$ of neighborhood sizes and interferences on the circle, respectively. Essentially, this is like having n random sensors thrown on the perimeter of the circle. Moreover, similar to Identity 1 we can show that $\sum_i E[N_C(i)] = \sum_j E[I_C(j)]$. Also, as in the proof of Lemma 7, we see easily that $E[N_C(i)] \in O(1)$, for all i. Since the random variables $I(j)$ are identically distributed we have that $\sum_j E[I_C(j)] = nE[I_C(j)] \in O(n)$. Therefore, $E[I_C(i)] \in O(1)$, for all i. Now observe that the circle experiment is the same as the interval experiment: the only sensors for which the neighborhood size and interference change is at the endpoints of the interval. Moreover, with high probability no two sensors can be at a distance higher than $\Omega(\ln n/n)$. Therefore with high probability, for all i we have that $I(i) \leq I_C(i)$ and consequently $E[I(i)] \leq E[I_C(i)]$. Therefore we have that $E[I(i)] \in O(1)$ for all i. This completes the proof of Theorem 1.

2.2 Probability Bound on Maximum Interference

Our bound on the probability of the maximum interference exceeding $\ln n$ follows from a similar bound on the size of a sensor's neighborhood.

Lemma 8. *For any constant $c > 1$ we have that*

$$\Pr[\max_i N(i) > 2c \ln n] \leq \frac{1}{n^{c-1}}.$$

Proof. Observe that if the event $\max_i N^+(i) > c \ln n$ holds then there exists an i such that $N^+(i) > c \ln n$. In turn, this last event implies that i has $c \ln n$ neighbors to its right, i.e., there exists a j which is i's neighbor and $j > i + c \ln n$. However, it is easy to show using Lemma 4 we have that the event $X_{i:n} - X_{i-1:n} > X_{i+2c \ln n:n} - X_{i:n}$ holds with probability less than $1/2n^c$. It follows that

$$\Pr[\max_i N^+(i) > 2c \ln n] \leq \frac{n}{n^c} = \frac{1}{2n^{c-1}}.$$

A similar argument will work for

$$\Pr[\max_i N^-(i) > 2c \ln n] \leq \frac{n}{n^c} = \frac{1}{2n^{c-1}}.$$

Therefore the lemma is proved.

We can now prove the desired result.

Theorem 2. *For any constant $c > 1$ we have that*

$$\Pr[I = \max_i I(i) > 4c \ln n] \leq \frac{1}{n^{c-1}}.$$

Proof. Using Lemma 8 as well as the easily proved fact $\max_i I(i) \leq 2 \max_j N(j)$, we see that

$$\Pr[\max_i I(i) > 4c \ln n] \leq \Pr[\max_j N(j) > 2c \ln n] \leq \frac{1}{n^{c-1}},$$

which proves the theorem.

3 Expected Maximum Interference

Clearly $E(I) = \Omega(1)$ and it is straightforward to show using Theorem 2 that $E(I) = O(\ln n)$. We now show tighter bounds on the expected maximum interference.

Theorem 3. *The expected maximum interference satisfies* $E[I] \in \Omega(\ln \ln n)$.

Proof. In this proof we use the equivalent Poisson model which we briefly discuss here. Consider a homogeneous Poisson process $\{N(t) : t \geq 0\}$ with parameter $\lambda > 0$. Let $0 = \Delta_0 < \Delta_1 < \cdots$. denote the successive arrival times of events of the process and for $j \geq 1$ let $T_j = \Delta_j - \Delta_{j-1}$ be the interarrival times (spacings) of the process. It is well-known that when conditioned under $N(t) = n, t > 0$, the random variables $0 \leq T_1 \leq \cdots \leq T_n \leq t$ are distributed as the order statistics of a sample of n observations taken from the uniform distribution on $[0, t]$. This is known to represent the most natural relationship between the Poisson process, random points on a line, and the uniform distribution of random points on an interval. (E.g., see [13] and [1]). Thus in the sequel we assume that we have a half-open (to the right) interval and sensors arrive with Poisson distribution and arrival rate $1/n$.

The probability that the interarrival time T between two consecutive sensors is $> r$ is e^{-rn}. In particular, elementary calculations show that

$$\Pr[a < T < b] = e^{-an} - e^{-bn}. \tag{5}$$

We will show that for each sensor the interference produced by sensors located to its right is at least $\ln \ln n$ with reasonable probability. To this effect we prove the following lemma.

Lemma 9. *Let* $T_j, T_{j+1}, \ldots, T_{j+k}$ *be the* $j, j+1, \ldots, j+k$ *interarrival times, respectively, of the Poisson process and consider the events*

$$A_{j,k} : \ (\forall i \leq k) \left(\frac{e^i}{n} < T_{j+i} < \frac{e^{i+1}}{n} \right).$$

Then

$$\Pr[A_{j,k}] \geq e^{-e^2 \frac{e^k}{e-1}}$$

Proof. To prove the lemma we use the independence of the interarrival times to conclude that

$$\begin{aligned}\Pr[A_{j,k}] &= \prod_{i=1}^{k} \Pr\left[\frac{e^i}{n} < T_{j+i} < \frac{e^{i+1}}{n}\right] \\ &= \prod_{i=1}^{k} \left(e^{-e^i} - e^{-e^{i+1}}\right) \\ &\geq \prod_{i=1}^{k} e^{-e^{i+1}} \\ &\geq e^{-e^2 \frac{e^k}{e-1}}\end{aligned}$$

which proves Lemma 9.

As a consequence of Lemma 9 we see that

$$\Pr\left[A_{j,k}\right] \geq \frac{1}{\sqrt{n}}, \tag{6}$$

for $k \leq \ln\ln n + \ln\left(\frac{e-1}{2e^2}\right)$. Fix $k = \left\lfloor \ln\ln n + \ln\left(\frac{e-1}{2e^2}\right)\right\rfloor$. We are now ready to prove Theorem 3.

Observe that the events $A_{j,k}$ and $A_{j',k}$ are independent, for $|j' - j| \geq k$. Let A_k be the event that at least one of the events $A_{1,k}, A_{1+\sqrt{n},k}, A_{1+2\sqrt{n},k}, \ldots$ is valid. It is easy to see

$$\begin{aligned}\Pr[A_k] &= 1 - \Pr[\neg A_k] \\ &= 1 - \Pr\left[\bigcap_u \neg A_{1+u\sqrt{n},k}\right] \\ &\geq 1 - \left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}}.\end{aligned}$$

Hence $\Pr[A_k] \geq 1 - \frac{1}{e}$.

Now it is possible to give a lower bound on the expected value of the maximum interference I. Indeed, since the occurrence of the event $A_{i,k}$ implies that the i-th sensor has interference at least k we see that

$$\begin{aligned}E[I] &= \sum_u \Pr[I > u] \\ &\geq \sum_{u \leq k} \Pr[I > u] \\ &\geq k \Pr[I > k] \\ &\geq k \Pr[A_k] \\ &\geq k\left(1 - \frac{1}{e}\right)\end{aligned}$$

This completes the proof of Theorem 3.

Finally we prove an upper bound on $E[I]$ better than that implied by Theorem 2.

Theorem 4. *For any $\epsilon > 0$ the expected maximum interference satisfies $E[I] = O((\ln n)^{1/2+\epsilon})$.*

Proof. Note that the result follows for $\epsilon \geq 1/2$ by Theorem 2. Throughout this proof we refer to the $O(n^2)$ intervals delimited by the n sensors as "discrete intervals". Recall from Lemma 2 that with high probability there is no gap (discrete interval containing no sensors) of size $c \ln n/n$, where c is an appropriately chosen constant. As a consequence, all sensors interfering with a given sensor must be at a distance $O(\ln n)$, with high probability. Note that the expected number of sensors in an interval Δ is equal to $n|\Delta|$, where $|\Delta|$ is the length of Δ.

Now we proceed with proving the main assertion of the theorem. The proof is by contradiction. Assume on the contrary that for some $\epsilon > 0$, $E[I] \geq (\ln n)^{1/2+\epsilon}$. We prove the following claim.

Claim 1. *There exists a sensor and a $c > 0$ such that for any interval of size at least $c \ln n/n$ centered at that sensor the sum of interferences of all sensors in the interval is $\Omega((\ln n)^{1+2\epsilon})$, with probability at least $1/(24(\ln n)^{1/2-\epsilon})$.*

Proof. Let $1/2 > \epsilon > 0$ be such that $E[I] \geq (\ln n)^{1/2+\epsilon}$. From Lemma 6 we get that for all i, $\Pr[I(i) > 12 \ln n] \leq 24 \ln n/n^4$. Therefore, for sufficiently large n, $\Pr[I = \max_i I(i) > 12 \ln n] \leq 1/n^2$. We see from the above that

$$\begin{aligned}
(\ln n)^{1/2+\epsilon} &\leq E[I] \\
&= \sum_v \Pr[I > v] \\
&= \sum_{v \leq 12 \ln n} \Pr[I > v] + o(1) \\
&= \sum_{v < (\ln n)^{1/2+\epsilon}/2} \Pr[I > v] + \sum_{(\ln n)^{1/2+\epsilon}/2 \leq v \leq 12 \ln n} \Pr[I > v] + o(1) \\
&\leq (\ln n)^{1/2+\epsilon}/2 + \sum_{(\ln n)^{1/2+\epsilon}/2 \leq v \leq 12 \ln n} \Pr[I > v] + o(1).
\end{aligned}$$

It follows that

$$(\ln n)^{1/2+\epsilon}/2 \leq \sum_{(\ln n)^{1/2+\epsilon}/2 \leq v \leq 12 \ln n} \Pr[I > v] + o(1).$$

This implies that the maximum interference satisfies $I > (1/2)(\ln n)^{1/2+\epsilon}$, with probability at least $1/(24(\ln n)^{1/2-\epsilon})$. However, if $I > (1/2)(\ln n)^{1/2+\epsilon}$ then there is a sensor, say i_0, with interference $\ell > (1/2)(\ln n)^{1/2+\epsilon}$. Hence, there are sensors with interference $\ell - 1, \ell - 2, \ldots, 1$, respectively, and all these sensors will lie within an interval of length $c \ln n/n$ centered at sensor i_0, with high probability.

Clearly in any interval of length $c \ln n/n$ containing this interval, the sum of interferences of all sensors will be at least $\ell(\ell-1)/2$ (which is in $\Omega((\ln n)^{1+2\epsilon})$) with probability at least $1/(24(\ln n)^{1/2-\epsilon})$. This proves the claim.

The next claim is an application of Chernoff bounds.

Claim 2. *There exists a $c > 0$ such that for any interval of length at least $c \ln n/n$ centered at any sensor, the sum of interferences of all sensors in this interval is $O(\ln n)$, with probability greater than $1 - 1/n$.*

Proof. Let c be a sufficiently large constant and consider an interval Δ of length $3c \ln n/n$. Further, let J be the interval centered at the midpoint of Δ and of length $c \ln n/n$. Let the random variable X_i indicate with 1 that the i-th sensor is in the interval J and is 0 otherwise. Using Chernoff bounds we see that

$$\Pr\left[\sum_{i=1}^{n} X_i > (1+\delta)\mu\right] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu},$$

where $\mu = 3c \ln n$ and $\delta > 0$ is chosen in such a way that $\frac{e^{\delta}}{(1+\delta)^{1+\delta}}$ is a constant < 1. It follows that for some constant $c' > 0$ the number of interfering sensors in the interval J will be at most $c' \ln n$, with probability, at least $1 - 1/n^2$. By Theorem 1, for each i the expected interference of a given sensor i is in $O(1)$. It follows that the sum of expected interferences of all sensors in the interval J will be in $O(\ln n)$, with high probability. Moreover there are at most $O(n)/\ln n$ intervals of length $O(\ln n)/n$ (one interval centered at a sensor, for each sensor). Hence, for all intervals J of length $O(\ln n)/n$ centered around sensors, the sum of expected interferences of all sensors in the interval J will be in $O(\ln n)$, with probability at least $1 - 1/n$. This proves the claim.

Now we can derive a contradiction. By Claim 2 and Boole's inequality, the probability that there is an interval of length $O(\ln n)/n$ with sum of interferences not in $O(\ln n)$ is strictly less than $1/n$. But this contradicts Claim 1 since

$$\begin{aligned}\frac{1}{24(\ln n)^{1/2-\epsilon}} &\geq \Pr\left[\exists J \left(\sum_{i\in J} I(i) \in \Omega((\ln n)^{1+2\epsilon})\right)\right] \\ &\geq \Pr\left[\exists J \left(\sum_{i\in J} I(i) \notin O(\ln n)\right)\right] \geq \frac{1}{n}.\end{aligned}$$

This proves the theorem.

4 Conclusion

In this paper we have investigated the receiver interference for a set of random sensors on a line (also known as the highway model) and proved upper and lower on the value of the expected maximum interference. Aside from tightening the

bounds, it would be interesting to look at probability distributions other than the uniform for the arrangement of sensors. Finally, bounds for the two dimensional case would be quite interesting. Unlike the one-dimensional case studied here whereby sensors where assigned as range the maximum distance between their two neighbors, an analysis of the two-dimensional case must be preceded by an assignment of sensor ranges, for example the maximum length of an edge in the minimum spanning tree.

Acknowledgments

We thank the anonymous referees for many helpful suggestions. This research was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC) and Mathematics of Information Technology and Complex Systems (MITACS).

References

1. Abay, A.: Extremes of Interarrival Times of a Poisson Process under Conditioning. Applicationes Mathematicae 23(1), 73–82 (1995)
2. Arnold, B.C., Balakrishnan, N., Nagaraja, H.N.: A first course in order statistics. John Wiley & Sons, Chichester (1992)
3. Bilo, D., Proletti, G.: On the complexity of minimizing interference in ad hoc and sensor networks. Theoretical Computer Science 402, 43–55 (2008)
4. Buchin, K.: Minimizing the maximum interference is hard, arXiv:0802.2134 (February 2008)
5. Burkhart, M., Wattenhofer, R., Zollinger, A.: Does topology control reduce interference? In: Proceedings of the 5th ACM International Symposium on Mobile ad hoc Networking and Computing, pp. 9–19. ACM, New York (2004)
6. Darling, D.A.: On a class of problems related to the random division of an interval. Ann. Math. Statist. 24, 239–253 (1953)
7. Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Transactions on Information Theory 46(2), 388–404 (2000)
8. Halldórsson, M.M., Tokuyama, T.: Minimizing interference of a wireless ad-hoc network in a plane. Theoretical Computer Science 402(1), 29–42 (2008)
9. Jain, K., Padhye, J., Padmanabhan, V.N., Qiu, L.: Impact of Interference on Multi-Hop Wireless Network Performance. Wireless Networks 11(4), 471–487 (2005)
10. Locher, T., von Rickenbach, P., Wattenhofer, R.: Sensor Networks Continue to Puzzle: Selected Open Problems. In: Rao, S., Chatterjee, M., Jayanti, P., Murthy, C.S.R., Saha, S.K. (eds.) ICDCN 2008. LNCS, vol. 4904, p. 25. Springer, Heidelberg (2008)
11. Moscibroda, T., Wattenhofer, R.: Minimizing interference in ad hoc and sensor networks. In: Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing, pp. 24–33. ACM, New York (2005)
12. Penrose, M.D.: The longest edge of the random minimal spanning tree. Annals of Applied Probability 7, 340–361 (1997)
13. Pyke, R.: Spacings. Journal of the Royal Statistical Society. Series B (Methodological) 27(3), 395–449 (1965)
14. von Rickenbach, P., Schmid, S., Wattenhofer, R., Zollinger, A.: A Robust Interference Model for Wireless Ad-Hoc Networks. In: Proc. 5th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad-Hoc and Sensor Networks, WMAN (2005)

Multipath Spanners

Cyril Gavoille[1,⋆], Quentin Godfroy[1,⋆], and Laurent Viennot[2,⋆⋆]

[1] University of Bordeaux, LaBRI
[2] INRIA, University Paris 7, LIAFA

Abstract. This paper concerns graph spanners that approximate multipaths between pair of vertices of an undirected graphs with n vertices. Classically, a spanner H of stretch s for a graph G is a spanning subgraph such that the distance in H between any two vertices is at most s times the distance in G. We study in this paper spanners that approximate short cycles, and more generally p edge-disjoint paths with $p > 1$, between any pair of vertices.

For every unweighted graph G, we construct a 2-multipath 3-spanner of $O(n^{3/2})$ edges. In other words, for any two vertices u, v of G, the length of the shortest cycle (with no edge replication) traversing u, v in the spanner is at most thrice the length of the shortest one in G. This construction is shown to be optimal in term of stretch and of size. In a second construction, we produce a 2-multipath $(2, 8)$-spanner of $O(n^{3/2})$ edges, i.e., the length of the shortest cycle traversing any two vertices have length at most twice the shortest length in G plus eight. For arbitrary p, we observe that, for each integer $k \geqslant 1$, every weighted graph has a p-multipath $p(2k-1)$-spanner with $O(pn^{1+1/k})$ edges, leaving open the question whether, with similar size, the stretch of the spanner can be reduced to $2k-1$ for all $p > 1$.

Keywords: spanner, multipath.

1 Introduction

This paper concerns the computation of sparse spanners for the multipath graph metric. We call *graph metric* a function δ that associates a metric δ_G with the vertex-set of a given graph G. A graph metric δ is *non-increasing* when distances can only decrease when adding edges. In other words, δ is non-increasing when $H \subseteq G$ implies $\delta_H \geqslant \delta_G$. (Here $H \subseteq G$ stands for H is a subgraph[1] of G, and $\delta_H \geqslant \delta_G$ stands for $\delta_H(u, v) \geqslant \delta_G(u, v)$ for all[2] u, v.) The *graph distance* d is the most classical graph metric: given an weighted undirected graph G, $d_G(u, v)$ is defined as the cost of a shortest path between u and v.

⋆ Supported by the ANR-project "ALADDIN", the équipe-projet INRIA "CÉPAGE", and the French-Israeli "Multi-Computing" project. The first author is Member of the "Insitut Universitaire de France".

⋆⋆ Supported by the ANR-project "ALADDIN", and the équipe-projet INRIA "GANG".

[1] I.e., $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

[2] For convenience, we set $\delta_H(u, v) = \infty$ if u or v is not in $V(H)$.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 211–223, 2010.

The notion of spanner is usually defined for the graph distance but it can be formulated for any non-increasing graph metric δ: an *f-spanner* (for δ) of a graph G is a subgraph $H \subseteq G$ such that $\delta_H(u,v) \leqslant f(\delta_G(u,v))$ for all $u,v \in V(G)$ where f is a stretch function satisfying $f(d) \geqslant d$ for $d \geqslant 0$. As δ is non-increasing, H must satisfy $\delta_G(u,v) \leqslant \delta_H(u,v) \leqslant f(\delta_G(u,v))$ for all u,v. Intuitively, an f-spanner H approximates the distances of G using possibly fewer edges. A rich literature studies the trade-off between the sparsity of H and the tightness of the stretch function f.

In this paper, we focus on the notion of spanner for the multipath graph metric. Given an integral value $p \geqslant 1$, the *p-multipath graph metric d^p* is defined as follows: given a weighted undirected graph G, $d^p_G(u,v)$ is the minimum weight sum of the edges of a set of p edge-disjoint paths joining u and v in G. The value $d^p_G(u,v)$ can be determined in polynomial time by computing a minimum-cost flow[3] of value p between source u and sink v. For $p = 1$, we fall back on the graph distance: $d^1_G(u,v) = d_G(u,v)$. This introduces a new notion of spanner that we call *multipath spanner*.

1.1 Motivation

Our interest to the multipath graph metric stems from the need for multipath routing in networks. Using multiple paths between a pair of nodes is an obvious way to aggregate bandwidth. Additionally, a classical approach to quickly overcome link failures consists in pre-computing fail-over paths which are link-wise disjoint from primary paths [18,23,22]. Multipath routing can be used for traffic load balancing and for minimizing delays [32,15]. Multipath routing has been extensively studied in ad hoc networks for load balancing, fault-tolerance, higher aggregate bandwidth, diversity coding, minimizing energy consumption (see [21] for a quick overview). Heuristics have been proposed to provide disjoint routes [22,19] in on-demand protocols. There is a wide variety of optimization requirements when using several paths between pairs of nodes. However, using edge-disjoint or vertex-disjoint paths is a recurrent concern in optimizing routing in networks. Using disjoint paths is a dissertation subject in itself [17] and has many problem variants.

Considering only a subset of links is a practical concern in link state routing in ad hoc networks [16]. This raises the problem of computing spanners for the multipath graph metric. As node mobility results in link failures, having an edge-disjoint multipath between two nodes reduces the probability of disconnection. Additionally, spanners are a key ingredient in the design of compact routing schemes [25,30]. Designing multipath spanners is thus a first step toward multipath compact routing.

Another reason for considering edge-disjoint paths rather than vertex-disjoint paths is that the resulting distance is a metric. This is not the case with vertex-disjoint paths. More specifically, if we define $d^{*p}_G(u,v)$ as the minimum cost of a

[3] To avoid using an edge in both directions, we apply a standard reduction to digraphs: each undirected edge xy is replaced by the dipath $x \to x' \to y' \to y \to x' \to y' \to x$.

set of p internally vertex-disjoint paths joining u and v, the triangle inequality may be violated. For $p \geqslant 2$, one can easily build a graph G where $d^{*p}_G(u,v)$ and $d^{*p}_G(v,w)$ are finite whereas $d^{*p}_G(u,w)$ is not[4]. The proof that d^p_G satisfies the triangle inequality is given in Proposition 1 (see Section 2.1). Choosing the weight sum as the cost of a set of p edge-disjoint paths follows again the same rationale. d^p_G is the most natural metric generalizing d_G in the context of multipath routing.

1.2 Related Work

The concept of spanner has been mostly studied for the graph distance as introduced by Peleg and Schäffer [24], and generalized to weighted graphs in [3]. There is an abundant literature on graph distance spanners which is surveyed, e.g., by Pettie in [26]. It is well-known that, for each integer $k \geqslant 1$, every weighted graph with n vertices has an f-spanner of $O(n^{1+1/k})$ edges with stretch function $f(d) = (2k-1)d$. This can be proved using a greedy construction based on a simple modification of Kruskall's algorithm[5], and this size-stretch tradeoff is conjectured to be tight. However, other tradeoffs exist, in particular for unweighted graphs and for stretch functions on the form $f(d) = \alpha d + \beta$. For $\alpha = 1$, the size is $O(n^{3/2})$ with $\beta = 2$ [1], and $O(n^{4/3})$ [4] with $\beta = 6$. There are also constructions for $\alpha = 1 + \varepsilon$ and arbitrary small $\varepsilon > 0$. The term $\beta = \beta(\varepsilon, k)$ depends on ε and k, and the size is $O(\beta n^{1+1/k})$ [11,31]. For $\alpha = O(1)$ and $\beta = \tilde{O}(1)$, spanners of size[6] $O(n)$ exist [26]. Other stretch functions have been considered, e.g., $f(d) = d + O(d^{1-1/k})$ [26,31]. Distributed algorithms for constructing such spanners have been also developped in [7,8,10,12,27].

The concept of spanner has been extended to some other graph metrics. In weighted directed graphs, the one-way distance is not a graph metric. However, the *roundtrip distance*, defined as the one-way distance from u to v plus the one way distance from v to u, is a non-increasing graph metric. Roundtrip f-spanners of size $O(n^{3/2})$ exist for $f(d) = 3d$ [28], and the size is $\tilde{O}(n^{1+1/k})$ for $f(d) = (2^k - 1)d$ and for every $k > 1$ [6]. It is also proved that no such size-stretch tradeoff can exist if the usual one-way distance in directed graph is considered. Interestingly, these solutions lead to compact routing scheme in directed graphs.

Dragan and Yan [9] study the complexity of computing spanners approximating maximum flow between any two vertices. We observe that the inverse[7] of the maximum flow is a graph metric (it is even an ultrametric), and thus is captured by our framework.

A concept independent of spanner for a graph metric δ, is the one of fault-tolerant spanner. An f-spanner H is *t-fault-tolerant* if $\delta_{H \setminus F}(u,v) \leqslant f(\delta_{G \setminus F}(u,v))$ for all vertices u, v and for any set F of at most t faulty vertices and/or edges. Introduced in [20] for the graph distance and for geometric

[4] E.g., if v is a cut-vertex.

[5] Visit all the edges of G by increasing order of their weights, and add the edge (x,y) to the current spanner H, initialized to the empty graph, only if $d_H(x,y) > (2k-1) \cdot d_G(x,y)$.

[6] Where $\tilde{O}(g(n))$ stands for $g(n) \cdot (\log n)^{O(1)}$.

[7] The flow from u to u is considered to be ∞.

graphs, namely the d-dimensional Euclidean complete graph, it has been recently generalized to arbitrary weighted graphs in [5]. Surprisingly, size-stretch tradeoffs are similar to the classic case where no-fault ($t = 0$) are tolerated. Note that t-fault-tolerant spanners preserve $(t+1)$-connectivity: if there exists $t+1$ disjoint paths in the graph from u to v, then the spanner also contains $t+1$ disjoint paths from u to v. However, the stretch guarantee is different from ours. In particular, we can build examples where the stretch guarantee is low with respect to fault tolerance whereas it is high when considering the multipath graph metric.

1.3 Our Contributions

Our results hold for undirected multi-edge graphs, and for the p-multipath graph metric d^p, for integer $p \geqslant 1$. Our contributions are the following:

1. We observe (Proposition 2) that every weighted graph has a p-multipath f-spanner of $O(pn^{1+1/k})$ edges, where $f(d) = p(2k-1)d$. This is done by an iterative construction of standard graph distance spanners.
2. The analysis of the stretch can be refined for unweighted graphs, and we show that the previous construction for $p = k = 2$ actually leads to a 2-multipath f-spanner of $O(n^{3/2})$ edges, where $f(d) = 3d$.
3. We also show that all the lower bounds for $p = 1$, i.e., for the standard graph distance, generalize to p-multipath distance for any $p > 1$. In particular, the size-stretch tradeoff of our second result is optimal.
4. Using a quite different approach, we show that 2-multipath f-spanner of size $\Phi \cdot n^{3/2} + n$ edges exists for $f(d) = 2d + 8W$, where W is the largest edge-weight of the graph and $\Phi \approx 1.61$ is the golden ratio.

It may be worth to mention that, as long as δ is a non-increasing graph metric, the greedy Kruskall's algorithm can be naively applied to produce sparse approximate skeletons. It suffices to construct, from G, the weighted complete graph K defined by $V(K) = V(G)$ and the weight $\delta_G(u, v)$ assigned to the edge (u, v). An f-spanner H of $O(n^{1+1/k})$ edges with $f(d) = (2k-1)d$ can therefore be constructed. Unfortunately, H is not a spanner of the input graph G, it is only a subgraph of K. This general solution might be acceptable for emulator construction [31] where the output graph H is not required to be a subgraph.

A second observation is that, in spite of similarities between roundtrip and 2-multipath distances (in both cases a subgraph realizing the distance between any two vertices indeed consists of a shortest cycle), there is no reduction from 2-multipath to roundtrip spanners[8].

In the next section we give formal definitions and prove some important basic facts, including simple upper and lower bounds. In Section 3, we present the optimal 2-multipath f-spanner with $O(n^{3/2})$ edges with $f(d) = 3d$. In Section 4, we improve the stretch function to $f(d) = 2d + 8$. We give some open problems in Section 5.

[8] By simply directing the edges of G and applying an efficient roundtrip spanner to G, we may obtain a roundtrip from u to v using twice the same arc, say $u \to a \to b \to v \to a \to b \to u$, which is not an acceptable solution for a 2-multipath spanner.

2 Preliminaries

In this paper, we consider an undirected multi-edge weighted graph G with weight function ω. The *cost* of any subgraph H of G is the sum of the weights of its edges. It is denoted by $\omega(H) = \sum_{e \in E(H)} \omega(e)$. We set $\omega(H) = 0$ if $E(H) = \varnothing$.

2.1 Multipath Distance and Multipath Spanner

A *p-path* from a vertex u to a vertex v is a subgraph of G composed of p edge-disjoint paths from u to v. We define the *p-multipath distance* between two vertices u and v, denoted by $d^p_G(u, v)$, as the minimum cost of a p-path from u to v. We set $d^p_G(u, v) = \infty$ if there are no p edge-disjoint paths from u to v.

Indeed, d^p_G is a distance. It clearly satisfies $d^p_G(u, v) = 0$ if and only if $u = v$. Symmetry follows from the fact that G is undirected, and the triangle inequality comes from Proposition 1.

Proposition 1. *Let u, v, w be any triple of vertices of a multi-edge graph G. If A is a p-path from u to v, and B a p-path from v to w, then $A \cup B$ contains a p-path from u to w. In particular, $d^p_G(u, w) \leqslant d^p_G(u, v) + d^p_G(v, w)$.*

Proof. Let (U, W) be a minimum cut between $u \in U$ and $w \in W$ in the graph induced by $H = A \cup B$. Let t be the number of edges with an endpoint in U and the other in W. Consider a maximum flow on H with source u and sink v with unit capacity on each edge. From the min-cut max-flow theorem we derive that there are t edge-disjoint paths from u to w. If $t < p$, we conclude that u and v are both in U as there exists a p-path between them. The same argument can be used to show that $v, w \in W$. This is a contradiction, $U \cap W = \varnothing$. □

The notions of p-path and p-multipath distance extend the usual notions of path and distance which correspond to the case $p = 1$. We write d_G a short for d^1_G. We observe that the multipath distance does not extend to a vertex-disjoint version. Indeed, for each $p > 1$, the existence of p vertex-disjoint paths from u to v and of p vertex-disjoint paths from v to w does not imply there are p vertex-disjoint paths from u to w.

A subgraph H of G is a *p-multipath s-spanner* if $d^p_H(u, v) \leqslant s \cdot d^p_G(u, v)$ for all pairs of vertices u, v. The parameter s is called the *p-multipath stretch* of H. We also use the term *stretch*, instead of multipath stretch, when the context is clear. For $p = 1$, we fall back on the regular definition of s-spanner.

2.2 Iterative Spanners

A *p-iterative s-spanner* of G is a subgraph $H = \bigcup_{i=1}^{p} H_i$, where H_i is any 1-multipath s-spanner of $G \setminus \bigcup_{j<i} H_j$. We observe that the union of p such 1-multipath spanners is actually a p-multipath spanner.

Proposition 2. *For all integers $k, p \geqslant 1$, every multi-edge weighted graph with n vertices has a p-multipath $p(2k-1)$-spanner with less than $p \cdot n^{1+1/k}$ edges that can be constructed as a p-iterative $(2k-1)$-spanner.*

Proof. Let $H = \bigcup_{i=1}^{p} H_i$ be a p-iterative $(2k-1)$-spanner of G, where H_i is a $(2k-1)$-spanner of G with less than $n^{1+1/k}$ edges. Each spanner H_i does exist (cf. [3]). Hence, H has less than $p \cdot n^{1+1/k}$ edges.

We now prove that H is a p-multipath $p(2k-1)$-spanner. Let u, v be two vertices of G. If there is no p-path from u to v in G, then $d_G^p(u,v) = \infty$. In particular, $d_H^p(u,v) \leqslant p(2k-1) \cdot d_G^p(u,v)$. So, we assume there exists a p-path from u to v. Let P be any minimum-cost p-path from u to v in G. We have $\omega(P) = d_G^p(u,v)$. For an edge $e \notin H$, we denote by $H_i(e)$ the simple path which replaces the edge e of G in the i-th spanner member of H. Observe that, for each i, $\omega(H_i(e)) \leqslant (2k-1) \cdot \omega(e)$ because $e \in G \setminus H_i$ and H_i has stretch $2k-1$.

Given P and H, we define the subgraph F as follows:

$$F \;:=\; (P \cap H) \cup \bigcup_{e \in P \setminus H} \bigcup_{i=1}^{p} H_i(e) \;.$$

Clearly, $F \subseteq H$, since each edge $e \in P$ is either in H or is replaced by $H_i(e)$ for some i. Moreover, we have $\omega(F) \leqslant p(2k-1) \cdot \omega(P)$ because:

$$\begin{aligned}
\omega(F) &\leqslant \omega(P \cap H) + \sum_{e \in P \setminus H} \sum_{i=1}^{p} \omega(H_i(e)) \\
&\leqslant \sum_{e \in P \cap H} \omega(e) + \sum_{e \in P \setminus H} p(2k-1) \cdot \omega(e) \\
&\leqslant \sum_{e \in P} p(2k-1) \cdot \omega(e) = p(2k-1) \cdot \omega(P) \;.
\end{aligned}$$

Therefore, the stretch of H is at most $p(2k-1)$ as claimed.

We now show that F contains a p-path from u to v, and for that we shall use the min-cut max-flow theorem. Consider a cut $(X, \overline{X})$ with $u \in X$ and $v \in \overline{X}$. Since P is a p-path from u to v, there is a subset C of the cut of at least p edges of P which have one endpoint in X and the other in $\overline{X}$. Two cases are possible:

1. Every edge of C belongs to F: the cut in F is already at least p.
2. One edge of C does not belong to F: p paths where added in F in replacement for this edge, so the minimum cut is at least p.

Therefore, the minimum cut in F is at least p. By the min-cut max-flow theorem there is p edge-disjoint paths from u to v in F. It follows that F contains a p-path from u to v. This completes the proof. □

The rest of the paper studies how the $p(2k-1)$ stretch bound can be improved.

2.3 Lower Bounds

For all integers p, n and real $s > 1$, denote by $m_p(n,s)$ the largest integer such that there exists a multi-edge weighted graph with n vertices for which every p-multipath spanner of stretch $< s$ requires $m_p(n,s)$ edges.

The value of $m_p(n,s)$ provides a lower bound on the sparsity of p-multipath spanners of stretch $< s$. To illustrate this, consider for instance $p=1$ and $s=3$. It is known that $m_1(n,3)=\Omega(n^2)$, by considering the complete bipartite graph $B=K_{\lfloor n/2\rfloor,\lceil n/2\rceil}$. Since all cycles of B have length at least 4, every proper subgraph H contains two vertices x and y which are neighbors in B but at distance at least 3 in the spanner. Thus H is an s-spanner with $s \geqslant 3$. In other words, every s-spanner of B, with $s < 3$ contains all the edges of B that is $\Omega(n^2)$ edges.

Unfortunately, this argument does not transfer to p-multipath spanners whenever $p > 1$. Indeed, with the same graph B, we get $d^p_B(x,y) = 1+3(p-1)$. And, if (x,y) is removed from any candidate spanner H, we only get $d^p_H(x,y) = 3p$. Hence, the stretch for H so proved is only $d^p_H(x,y)/d^p_G(x,y) = 1+O(1/p)$. The argument transfers however if multi-edges are allowed.

Proposition 3. *For all integers $n, p \geqslant 1$ and real $s > 1$, $m_p(n,s) \geqslant m_1(n,s)$.*

In particular, under the Erdös-Simonovits [13,14] Conjecture[9] *which implies $m_1(n,2k+1) = \Omega(n^{1+1/k})$ for every integer $k \geqslant 1$ and proved for $k \in \{1,2,3,5\}$, there is a multi-edge unweighted graph with n vertices for which every p-multipath spanner with stretch $< 2k+1$ has $\Omega(n^{1+1/k})$ edges.*

Proof. Let G be an n-vertex graph with the minimum number of edges such that every spanner of stretch $< s$ has $m_1(n,s)$ edges. Let ω be the weight function of G. Clearly, G has $m_1(n,s)$ edges, since otherwise we could remove an edge of G. Observe also that any path between two neighbors x, y of G that does not use the edge (x,y) has length at least s, since otherwise we could remove it from G. In other words, $d_{G\setminus\{(x,y)\}}(x,y) \geqslant s \cdot \omega(x,y)$.

Let G_p be the graph obtained from G by adding, for each edge of G, $p-1$ extra multi-edges with same weight. We have $G_1 = G$, and G_p has $p \cdot m_p(n,s)$ edges. Let H be any p-multipath spanner of G_p with $< m_1(n,s)$ edges. There must exist two vertices x, y adjacent in G_p that are not adjacent in H. We have $d^p_{G_p}(x,y) = p \cdot \omega(x,y)$, and $d^p_H(x,y) \geqslant p \cdot d_{G\setminus\{(x,y)\}}(x,y) \geqslant p \cdot s \cdot \omega(x,y)$. We conclude that the p-multipath stretch of H is at least $d^p_{G_p}(x,y)/d^p_H(x,y) \geqslant s$.

In other words, every p-multipath spanner H of G_p with stretch $< s$ has $\geqslant m_1(n,s)$ edges, proving that $m_p(n,s) \geqslant m_1(n,s)$. □

We leave open the question of determining whether the same lower bound of $2k-1$ on the stretch applies if the graphs are restricted to be simple graphs only.

3 An Unweighted 2-Multipath 3-Spanner

In this section, we focus on unweighted 2-multipath 3-spanners. The lower bound of Proposition 3 tells us that $\Theta(n^{3/2})$ is the required size of any 2-multipath 3-spanner. However, the p-iterative $(2k-1)$-spanner given by Proposition 2 (with $p=k=2$) provides a 2-multipath spanner of stretch 6 only. In fact a finer analysis shows that the same construction yields a 2-multipath 3-spanner.

[9] It states that there are n-vetex graphs with $\Omega(n^{1+1/k})$ edges without cycles of length $\leqslant 2k$.

Theorem 1. *Every multi-edge unweighted graph with n vertices has a 2-multipath 3-spanner with less than $2n^{3/2}$ edges that can be constructed as a 2-iterative 3-spanner.*

It is obvious from the construction that a 2-iterative 3-spanner contains less than $2n^{3/2}$ edges. For the stretch, the long proof is divided in five lemmatas that are given in the full version of the paper.

4 A 2-Multipath (2,8)-Spanner

In this section, we construct a 2-multipath spanner with $O(n^{3/2})$ edges whose stretch is below 3 for (2-multipath) distances > 8. In the remaining, (α, β)-spanner stands for f-spanner of stretch function $f(d) = \alpha d + \beta$.

4.1 Multipath Spanning Trees

To prove the main result of this section we extend the notion of spanning tree.

A *p-multipath spanning tree of G* is a subgraph T of G with a distinguished vertex u, called the *root* of T, such that, for every vertex v of G, T contains a p-path from u to v. Moreover, T is a *p-shortest-path spanning tree* if $d^p_T(u, v) = d^p_G(u, v)$ for every vertex v. For $p = 1$, we come back to the standard notions of spanning tree and shortest-path spanning tree. Observe that T may not exist, for instance, if G is not 2-edge-connected.

Lemma 1. *Every n-vertex 2-edge-connected graph with a given vertex u has a 2-shortest-path spanning tree rooted at u with at most $2(n-1)$ edges constructible in polynomial time.*

Proof. We use the construction of [29] that can be extended to undirected graphs. The algorithm of [29] constructs 2-paths, each one of minimum cost, from every vertex to a fixed source of the graph. Roughly speaking, the construction results of two shortest-path spanning trees computed with Dijkstra's algorithm. The 2-paths (from every vertex v to the source) are reconstructed via a specific procedure. This latter can be analyzed so that the number of edges used in the 2-multipath tree is at most $2(n-1)$: all vertices, but the source, have two parents. □

The bound of $2(n-1)$ is tight because of the graph $K_{2,n-2}$. More generally, the number of edges in any p-multipath spanning tree must be, in the worst-case, at least $p(n-p)$, for every $p \leqslant n/2$. Indeed, every p-multipath spanning tree T must be p-edge-connected[10], and the graph $K_{p,n-p}$ is minimal for the p-edge-connectivity. Therefore, T contains all the edges of $K_{p,n-p}$, and there are $p(n-p)$ edges. Obviously, there are p-multipath spanning tree with less than $p(n-p)$ edges. Typically a subdivision of $K_{2,p}$ with n vertices has p-multipath spanning tree rooted at a degree-p vertex with a total of $n+p-2$ edges.

[10] By Proposition 1, there are two edge-disjoint paths between any two vertices of the p-multipath tree, through its root.

4.2 A Stretch-(2,8W) Spanner

Theorem 2. *Every multi-edge weighted graph with n vertices and largest edge-weight W has a 2-multipath $(2, 8W)$-spanner with less than $\Phi\, n^{3/2} + n$ edges, where $\Phi \approx 1.618$ is the golden ratio.*

Proof. Let denote by $B^p_H(u, r) = \{v \in V(H) : d^p_H(u, v) \leqslant r\}$ the p-multipath ball of radius r in H centrered at u, and denote by $N^p_H(u, r)$ the neighbors of u in H that are in $B^p_H(u, r)$. Note that for $p \geqslant 2$, some neighbor v of u might not be in $B^p_H(u, r)$ for every $r < \infty$: for instance if u and v are not in the same 2-edge-connected component. We denote by $\mathrm{SPT}^p_H(u)$ a p-shortest-path tree rooted at u spanning the 2-edge-connected component of H containing u, and having at most $2(|E(H)| - 1)$ edges. According to Lemma 1, such p-shortest path tree can be constructed.

Let G be a multi-edge weighted graph with n vertices and largest edge-weight W. We denote by ω its edge-weight function. The 2-multipath spanner H of G is constructed thanks to the following algorithm (see Algorithm 1).

1. For each edge e of G: if there are in G two other edges between the endpoints of e with weight at most $\omega(e)$, then $G := G \setminus \{e\}$
2. $H := (V(G), \varnothing)$ and $W := \max\{\omega(e) : e \in E(G)\}$
3. While there exists $u \in V(G)$ such that $|N^2_G(u, 4W)| \geqslant (\sqrt{5} - 1)\sqrt{n}$:
   (a) $H := H \cup \mathrm{SPT}^2_G(u)$
   (b) $G := G \setminus N^2_G(u, 4W)$
   (c) $W := \max\{\omega(e) : e \in E(G)\}$
4. $H := H \cup G$

Algorithm 1. A 2-multipath $(2, 8W)$-spanner algorithm

Size: Denote by G_3 and H_3 respectively the graphs G and H obtained after running the while-loop. Let b be the number of while-loops performed by the algorithm, and let $a = \sqrt{5} - 1$. Observe that $a^2 + 2a = 4$. From Lemma 1, the 2-shortest-path tree $\mathrm{SPT}^2_G(u)$ has at most $2(n - 1)$ edges. Hence, the size of H_3 is at most:

$$|E(H_3)| < 2b \cdot n \ .$$

The number of vertices of G_3 is at most $n - ab\sqrt{n}$, since at each loop, at least $a\sqrt{n}$ vertices are removed from G. Let G^1_3 be the graph induced by all the edges (u, v) of G_3 such that $v \in N^2_{G_3}(u, 4W_1)$, where W_1 is the maximum weight of an edge of the graph obtained after running Instruction 1. Let G^2_3 be the graph induced by the edges of $G_3 \setminus G^1_3$. The degree of each vertex u of G^1_3 is $|N^2_{G_3}(u, 4W_1)| - 1$ which is $< \lceil a\sqrt{n} \rceil - 1 \leqslant a\sqrt{n}$ because of the while-condition. Therefore, the size of G^1_3 is at most:

$$|E(G^1_3)| \leqslant \frac{1}{2} \sum_{u \in V(G_3)} a\sqrt{n} \ < \ \frac{1}{2}\left(n - ab\sqrt{n}\right) \cdot a\sqrt{n} < \frac{a}{2} \cdot n^{3/2} - \frac{a^2 b}{2} \cdot n \ .$$

Let S_3 be the graph obtained from G_3^2 where each multi-edge is replaced by a single unweighted edge. More formally, vertices u and v are adjacent in S_3 if and only if there is at least one edge between u and v in G_3^2. From Instruction 1, there is at most two edges between two adjacent vertices, so $|E(G_3^2)| \leqslant 2|E(S_3)|$.

Let us show that S_3 has no cycle of length $\leqslant 4$. Consider any edge (u, v) of S_3. Observe that $v \notin N^2_{G_3^2}(u, 4W_3)$, where W_3 is the maximum weight of an edge of G_3. Assume there is a path cycle of length at most 4 in S_3 going through (u, v). Then in G_3^2 there is a 2-path from u to v of cost at most $4W_3$. Contradiction: $v \notin N^2_{G_3^2}(u, 4W_3)$ implies $d^2_{G_3^2}(u, v) > 4W_3$.

It has been proved in [2] that every simple η-vertex μ-edge graph without cycle of length $\leqslant 2k$, must verify the Moore bound:

$$\eta \;\geqslant\; 1 + \delta \sum_{i=0}^{k-1} (\delta - 1)^i \;>\; (\delta - 1)^k$$

where $\delta = 2\mu/\eta$ is the average degree of the graph. This implies that $\mu < \frac{1}{2}(\eta^{1+1/k} + \eta)$.

We observe that S_3 is simple. It follows, for $k = 2$ and $\eta \leqslant n - ab\sqrt{n}$, that the size of G_3^2 is at most (twice the one of S_3):

$$|E(G_3^2)| \leqslant \left(n - ab\sqrt{n}\right)^{3/2} + n - ab\sqrt{n} < (n - ab\sqrt{n})\sqrt{n} + n = n^{3/2} + (1 - ab) \cdot n \,.$$

Overall, the total number of edges of the final spanner H is bounded by:

$$\begin{aligned} |E(H)| &\leqslant |E(H_3)| + |E(G_3^1)| + |E(G_3^2)| \\ &< \left(1 + \frac{a}{2}\right) \cdot n^{3/2} + \left(2b - \frac{a^2 b}{2} + 1 - ab\right) \cdot n \\ &= \left(1 + \frac{a}{2}\right) \cdot n^{3/2} + n \;=\; \frac{1 + \sqrt{5}}{2} \cdot n^{3/2} + n \;=\; \Phi\, n^{3/2} + n \end{aligned}$$

because the term $2b - a^2 b/2 + 1 - ab = b/2 \cdot (4 - a^2 - 2a) + 1 = 1$. (Recall that, by the choice of a, $a^2 + 2a = 4$.)

Stretch: Let G_0 be the input graph G, before applying the algorithm. We first observe that we can restrict our attention to the stretch analysis of G_1 (instead of G_0), the graph obtained after applying Instruction 1.

Let H be a 2-multipath spanner for G_1. Consider two vertices u, v of H, and let A be a minimum-cost 2-path between u and v in H. A is composed of two edge-disjoint paths and is of minimum cost in H, so A traverses (at most) two edges with same endpoints having the smallest weight. These (possibly) two edges exist in G_0 and in G_1, therefore the 2-multipath stretch of H in G_0 or in G_1 is the same.

From the above observation, it suffices to prove that H is a 2-multipath $(2, 8W_1)$-spanner of G_1, where $W_1 \leqslant W_0$ is the maximum weight of an edge of G_1.

Let x, y be any two vertices of G_1, and A be a minimum-cost 2-path between x and y in G_1. Let $d = d^2_{G_1}(x, y) = \omega(A)$. If all the edges of A are in H, then

$d^2_H(x,y) = d^2_{G_1}(x,y) = d$, and the stretch is $(1,0)$. So, assume that $A \not\subset H$. Let u be the first vertex selected in the while-loop such that $N^2_G(u, 4W)$ intersects A, so that Instruction 3(b) removes at least one edge of A. Let G, H be the graphs at the time u is selected, but before running Instruction 3(a) and 3(b). Let $v \in N^2_G(u, 4W) \cap A$, and B a minimum-cost 2-path from u and v in G. By definition of $N^2_G(u, 4W)$, $d^2_G(u,v) = \omega(B) \leqslant 4W$. Let $T = \mathrm{SPT}^2_G(u)$.

An important observation is that u, x, y are in the same 2-edge-connected component of G. This comes from the fact that every 2-path is a 2-edge-connected subgraph[11]. So, A and B are 2-edge-connected, and $A \cup B$ as well, since A intersects B (in v).

Using the triangle inequality (Proposition 1) between x and y in H, we have $d^2_H(x,y) \leqslant d^2_H(u,x) + d^2_H(u,y)$. By construction of H and T, $d^2_H(u,x) = d^2_T(u,x) = d^2_G(u,x) \leqslant \omega(A \cup B)$ since we have seen that $u, x \in A \cup B$ that is 2-edge-connected. Thus, $d^2_H(u,x) \leqslant \omega(A) + \omega(B) \leqslant d + 4W_1$. Similarly, $d^2_H(u,y) \leqslant d + 4W_1$. Therefore, $d^2_H(x,y) \leqslant 2d + 4W_1$. The subgraph H is a 2-multipath $(2, 8W_1)$-spanner, completing the proof. □

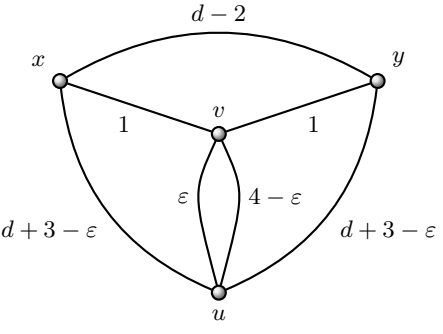


Fig. 1. A weighted graph G with $d^2_G(x,y) = d$ showing that the stretch analysis in the proof of Theorem 2 is tight. The 2-shortest-path tree rooted at u spans all the edges but (x,y). We have $d^2_T(u,x) = d^2_T(u,y) = d + 4$, and $d^2_T(x,y) = 2d + 8 - 2\varepsilon$.

5 Conclusion

We have introduced multipath spanners, some subgraphs that approximate the cost of p edge-disjoint paths between any two vertices of a graph. The usual notion of spanner is obtained for $p = 1$. This new notion leaves open several questions. We propose some of them:

– Is it true that every weighted graph has a p-multipath $(2k-1)$-spanner with $O(pn^{1+1/k})$ edges?

[11] This observation becomes wrong whenever p-paths with $p > 2$ are considered.

- Let $s < 2k + 1$. Does the bound $\Omega(n^{1+1/k})$ on the size of any p-multipath s-spanner hold for the class of n-vertex simple graphs (with no multi-edges)?
- Let $\varepsilon \in (0, 1]$. Is there a 2-multipath $(2 - \varepsilon, O(W))$-spanner with $o(n^2)$ edges for every graph of largest edge-weight W? and with $O(n^{3/2})$ edges? or even $O(n^{4/3})$ edges? And for $p > 2$?
- Prove, for $p > 2$, that every p-edge-connected graph has a p-shortest-path spanning tree with at most $p(n - 1)$ edges.

References

1. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J. on Computing 28, 1167–1181 (1999)
2. Alon, N., Hoory, S., Linial, N.: The Moore bound for irregular graphs. Graphs and Combinatorics 18, 53–57 (2002)
3. Althöfer, I., Das, G., Dobkin, D., Joseph, D.A., Soares, J.: On sparse spanners of weighted graphs. Discrete & Computational Geometry 9, 81–100 (1993)
4. Baswana, S., Kavitha, T., Mehlhorn, K., Pettie, S.: New constructions of (α, β)-spanners and purely additive spanners. In: 16th Symposium on Discrete Algorithms (SODA), January 2005, pp. 672–681. ACM-SIAM (2005)
5. Chechik, S., Langberg, M., Peleg, D., Roditty, L.: Fault-tolerant spanners for general graphs. In: 41stAnnual ACM Symposium on Theory of Computing (STOC), May 2009, pp. 435–444. ACM Press, New York (2009)
6. Cowen, L.J., Wagner, C.: Compact roundtrip routing in directed networks. In: 19th Annual ACM Symposium on Principles of Distributed Computing (PODC), July 2000, pp. 51–59. ACM Press, New York (2000)
7. Derbel, B., Gavoille, C., Peleg, D., Viennot, L.: On the locality of distributed sparse spanner construction. In: 27th Annual ACM Symposium on Principles of Distributed Computing (PODC), August 2008, pp. 273–282. ACM Press, New York (2008)
8. Derbel, B., Gavoille, C., Peleg, D., Viennot, L.: Local computation of nearly additive spanners. In: Keidar, I. (ed.) DISC 2009. LNCS, vol. 5805, pp. 176–190. Springer, Heidelberg (2009)
9. Dragan, F.F., Yan, C.: Network flow spanners. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 410–422. Springer, Heidelberg (2006)
10. Elkin, M.: Computing almost shortest paths. ACM Transactions on Algorithms 1, 283–323 (2005)
11. Elkin, M., Peleg, D.: $(1 + \epsilon, \beta)$-spanner constructions for general graphs. SIAM J. on Computing 33, 608–631 (2004)
12. Elkin, M., Zhang, J.: Efficient algorithms for constructing $(1 + \epsilon, \beta)$-spanners in the distributed and streaming models. Distributed Computing 18, 375–385 (2006)
13. Erdös, P.: Extremal problems in graph theory. In: Publ. House Cszechoslovak Acad. Sci., Prague, pp. 29–36 (1964)
14. Erdös, P., Simonovits, M.: Compactness results in extremal graph theory. Combinatorica 2, 275–288 (1982)
15. Gallager, R.G.: A minimum delay routing algorithm using distributed computation. IEEE Transactions on Communications (1977)
16. Jacquet, P., Viennot, L.: Remote spanners: what to know beyond neighbors. In: 23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS), May 2009. IEEE Computer Society Press, Los Alamitos (2009)

17. Kleinberg, J.: Approximation algorithms for disjoint paths problems, PhD thesis, Massachusetts Institute of Technology (1996)
18. Kushman, N., Kandula, S., Katabi, D., Maggs, B.M.: R-bgp: Staying connected in a connected world. In: 4th Symposium on Networked Systems Design and Implementation, NSDI (2007)
19. Lee, S., Gerla, M.: Split multipath routing with maximally disjoint paths in ad hoc networks. In: IEEE International Conference on Communications (ICC), vol. 10, pp. 3201–3205 (2001)
20. Levcopoulos, C., Narasimhan, G., Smid, M.: Efficient algorithms for constructing fault-tolerant geometric spanners. In: 30th Annual ACM Symposium on Theory of Computing (STOC), May 1998, pp. 186–195. ACM Press, New York (1998)
21. Mueller, S., Tsang, R.P., Ghosal, D.: Multipath routing in mobile ad hoc networks: Issues and challenges. In: Calzarossa, M.C., Gelenbe, E. (eds.) MASCOTS 2003. LNCS, vol. 2965, pp. 209–234. Springer, Heidelberg (2004)
22. Nasipuri, A., Castañeda, R., Das, S.R.: Performance of multipath routing for on-demand protocols in mobile ad hoc networks. Mobile Networks and Applications 6, 339–349 (2001)
23. Pan, P., Swallow, G., Atlas, A.: Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090 (Proposed Standard) (2005)
24. Peleg, D., Schäffer, A.A.: Graph spanners. J. of Graph Theory 13, 99–116 (1989)
25. Peleg, D., Ullman, J.D.: An optimal synchornizer for the hypercube. SIAM J. on Computing 18, 740–747 (1989)
26. Pettie, S.: Low distortion spanners. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 78–89. Springer, Heidelberg (2007)
27. Pettie, S.: Distributed algorithms for ultrasparse spanners and linear size skeletons. In: 27th Annual ACM Symposium on Principles of Distributed Computing (PODC), August 2008, pp. 253–262. ACM Press, New York (2008)
28. Roditty, L., Thorup, M., Zwick, U.: Roundtrip spanners and roundtrip routing in directed graphs. ACM Transactions on Algorithms 3, 29 (2008)
29. Suurballe, J.W., Tarjan, R.E.: A quick method for finding shortest pairs of disjoint paths. Networks 14, 325–336 (1984)
30. Thorup, M., Zwick, U.: Approximate distance oracles. J. of the ACM 52, 1–24 (2005)
31. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: 17th Symposium on Discrete Algorithms (SODA), January 2006, pp. 802–809. ACM-SIAM (2006)
32. Vutukury, S., Garcia-Luna-Aceves, J.J.: A simple approximation to minimum-delay routing. In: SIGCOMM, pp. 227–238 (1999)

Strong Orientations of Planar Graphs with Bounded Stretch Factor

Evangelos Kranakis[1,⋆], Oscar Morales Ponce[1,⋆⋆], and Ladislav Stacho[2,⋆⋆⋆]

[1] School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada
[2] Department of Mathematics, Simon Fraser University, 8888 University Drive, Burnaby, British Columbia, V5A 1S6, Canada

Abstract. We study the problem of orienting some edges of given planar graph such that the resulting subdigraph is strongly connected and spans all vertices of the graph. We are interested in orientations with minimum number of arcs and such that they produce a digraph with bounded stretch factor. Such orientations have applications into the problem of establishing strongly connected sensor network when sensors are equipped with directional antennae.

We present three constructions for such orientations. Let $G = (V, E)$ be a connected planar graph without cut edges and let $\Phi(G)$ be the degree of largest face in G. Our constructions are based on a face coloring, say with λ colors. First construction gives a strong orientation with at most $\left(2 - \frac{4\lambda-6}{\lambda(\lambda-1)}\right)|E|$ arcs and stretch factor at most $\Phi(G) - 1$. The second construction gives a strong orientation with at most $|E|$ arcs and stretch factor at most $(\Phi(G) - 1)^{\lceil \frac{\lambda+1}{2} \rceil}$. The third construction can be applied to planar graphs which are 3-edge connected. It uses a particular 6-face coloring and for any integer $k \geq 1$ produces a strong orientation with at most $\left(1 - \frac{k}{10(k+1)}\right)|E|$ arcs and stretch factor at most $\Phi^2(G)(\Phi(G) - 1)^{2k+4}$. These are worst-case upper bounds. In fact the stretch factors depend on the faces being traversed by a path.

Keywords and Phrases: Digraph, Directional Antennae, Planar, Sensors, Cut Edges, Spanner, Stretch Factor, Strongly Connected.

1 Introduction

Directional antennae are widely being used in wireless networks not only for reducing energy consumption and interference, but also for improving routing efficiency and security. Sensors rely on the use of antennae to configure and operate an ad hoc network. Essentially, two types of antennae are being used: *Omnidirectional* antennae which transmit the signal in all directions in the plane

⋆ Supported in part by NSERC and MITACS grants.
⋆⋆ Supported in part by CONACYT and NSERC grants.
⋆⋆⋆ Supported in part by NSERC grant.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 224–236, 2010.

and *directional* antennae which can transmit the signal towards a specific direction. Omnidirectional antennae usually incur more interference than directional antennae thus hampering nodes from receiving data from other transmitters and causing overall performance degradation of the sensor network. Sensor networks using directional antennae not only can have extended life-time since the consumption of energy in each antenna is proportional to the area covered by the transmitting antennae, but also using a small antenna spread prevents unwanted nodes from listening to the communication and therefore, improving the security of the network. Hence, it is desirable to reduce not only the range, but also the angle of an antenna.

There has been some recent research concerning the advantages of using directional antennae. For example, Gupta et al. [7] have shown that when n omnidirectional antennae are being used in an area of one unit square, the throughput per node is at most $O(W/\sqrt{n})$, where each antenna can transmit W bits per second over the common channels, regardless of the sensor placement. This can be contrasted with Yi et al. [13] which show that directional antennae provide an improvement on the throughput capacity by a factor of $2\pi/\sqrt{\alpha\beta}$, where α is the angle of transmission and β is the angle of reception. In fact, when α and β go to zero, the wireless network behaves like a wired network from the throughput point of view. Similarly, Kranakis et al. [9] study the energy consumption of networks of omnidirectional antennae and compare it to the consumption of networks of directional antennae.

Motivated from these issues, in this paper we study the problem of directing edges of an undirected (connected) planar graph in such a way that the resulting digraph spans all vertices, is strongly connected, has bounded stretch factor, and the number of arcs employed is minimized. Note that if the undirected graph is hamiltonian then a solution is to orient edges along a Hamilton cycle. This yields an orientation that is strongly connected and has the minimum possible arcs. However the stretch factor of such orientation is unbounded. On the other hand, one can orient every edge of an undirected graph in two oposite directions. This will result in an orientation that is strongly connected with stretch factor equal to one. However the number of arcs in such an orientation is largest possible. Hence we are looking for tradeoffs between these two approaches.

1.1 Notation and Preliminaries

A set of sensors with omnidirectional antennae is modeled as an (undirected) geometric graph whose vertices are points in the plane and edges are straight line segments representing the connectivity between two sensors. This contrasts with a set of sensors with directional antennae which is modeled as a directed geometric graph (digraph) where the direction on an arc represents the direction of the corresponding communication link. Geometric graphs are always associated with a planar embedding and so we will consider them as planar graphs (digraphs) in this paper and will speak of their set of vertices V, edges E, and faces F, respectively. Our graphs (digraphs) will not have loops and/or multiple

edges (arcs). Given two integers $a < b$, let $[a, b] = \{a, a + 1, \ldots, b\}$ denote the integer interval. A (face) λ-coloring $\Lambda : F \to [1, \lambda]$ of a planar graph $G(V, E, F)$ is an assignment Λ of λ colors to faces of G such that adjacent faces, i.e. faces sharing a common edge, are assigned distinct colors.

Let G be a graph. An orientation $\boldsymbol{G}$ of G is a digraph obtained from G by orienting every edge of G in at least one direction. Observe that in our definition an edge can be bidirectional. As usual, we denote with (u, v) the arc from u to v, whereas $\{u, v\}$ denotes an undirected edge between u and v. Let $d(f)$ denote the degree of the face f in a planar graph (digraph) G i.e. the number of edges surrounding f. Finally let $\Phi(G)$ be the maximum degree of a face in G, i.e. $\Phi(G) = \max_{f \in F} d(f)$.

The stretch factor or spanning ratio of a strongly connected orientation $\boldsymbol{G}$ is the minimum value t such that for every ordered pair of vertices u and v and for every path from u to v in G there exists a directed path from u to v in $\boldsymbol{G}$ of length at most t times the length of the original path.

1.2 Related Work

Caragiannis et al. [3] were the first to propose the problem of orienting the antennae of a set of sensors in the plane and compared the range used to the maximum edge length of the minimum spanning tree on the set of sensors. They proposed a polynomial time algorithm in which the sensors use the optimal possible range to maintain connectivity in order to construct a strongly connected graph when the antennae spread is at least $8\pi/5$. In addition, they studied the case when the antennae spread α is in the interval $[\pi, 8\pi/5)$ and gave an algorithm which extends the antenna range to at most $2\sin(\pi - \alpha/2)$ times the minimum range required so as to maintain connectivity. They also showed that if the antenna spread is at most $2\pi/3$ the problem of constructing a strongly connected graph is NP-hard. In fact, when the angles of the antennae are equal to 0 this last problem is equivalent to the bottleneck traveling salesman problem [11] for which an approximation with radius 2 times the optimal is given in [11].

Bhattacharya et al. [1] extend the work in [3] and give results for more than one antenna per sensor. A more comprehensive study is provided by Dobrev et al. [5]. They consider the previously mentioned model of Caragiannis et al. [3] in order to study the optimal antennae range required when sensors are equipped with more than one antenna having spread 0. They show that the required range is $\sqrt{3}$ times the optimal for two antennae, $\sqrt{2}$ times the optimal for three antennae and $2\sin(\pi/5)$ times the optimal for four antennae.The problem considered in the present paper differs from the problems studied in [3], [1] and [5] in that we do not alter (increase) the sensor range, rather we work with given undirected graph (unit disk graph or its planar spanner).

Similar problem that has been addressed in the literature is one that studies connectivity requirements an undirected graph that will guarantee highest edge connectivity of its orientation, c.f. [6] and [10].

1.3 Contributions

Let $G = (V, E)$ be a connected planar graph without cut edges and let Λ be a face coloring, say with λ colors. We present three polynomial constructions for orientations of G. First construction (presented in Section 2) gives a strong orientation with at most $\left(2 - \frac{4\lambda-6}{\lambda(\lambda-1)}\right)|E|$ arcs and the stretch factor at most $\Phi(G) - 1$. The second construction (presented in Section 3) gives a strong orientation with at most $|E|$ arcs and the stretch factor at most $(\Phi(G) - 1)^{\lceil \frac{\lambda+1}{2} \rceil}$. The third construction (presented in Section 4) can be applied to planar graphs which are 3-edge connected. It uses a particular 6-face coloring and for any integer $k \geq 1$ produces a strong orientation with at most $\left(1 - \frac{k}{10(k+1)}\right)|E|$ arcs and the stretch factor at most $\Phi^2(G)(\Phi(G) - 1)^{2k+4}$.

2 Orientations with More than $|E|$ Arcs

Let $G(V, E, F)$ be a simple planar geometric graph. We want to orient edges in E so that the resulting digraph is strongly connected. A trivial algorithm is to orient each edge in E in both directions. In this case, the number of arcs is $2|E|$ and the stretch factor is 1. In this section we prove that it is possible to orient less than $2|E|$ edges of G and still maintain bounded stretch factor. Our approach is based on a λ-coloring of faces in F, for some integer λ. The idea of employing face coloring was used in [14] to construct directed cycles. Intuitively we give directions to edges depending on the color of their incident faces.

Theorem 1. *Let $G(V, E, F)$ be a planar geometric graph which is 2-edge connected. Suppose G has a face λ-coloring for some integer λ. There exists a strongly connected orientation $\boldsymbol{G}$ with at most*

$$\left(2 - \frac{4\lambda - 6}{\lambda(\lambda - 1)}\right) \cdot |E| \tag{1}$$

arcs, so that its stretch factor is at most $\Phi(G) - 1$.

Before giving the proof, we introduce some useful ideas and preliminary results that will be required.

Consider a planar geometric graph $G(V, E, F)$ and a face λ-coloring Λ of G with colors $\{1, 2, \ldots, \lambda\}$. Let $\boldsymbol{G}$ be the orientation resulting from giving two opposite directions to each edge in E. For each arc (u, v), we define $L_{u,v}$ as the face which is incident to $\{u, v\}$ on the left of (u, v), and similarly $R_{u,v}$ as the face which is incident to $\{u, v\}$ on the right of (u, v). Observe that for given embedding of G, $L_{u,v}$ and $R_{u,v}$ are well defined. Since G has no cut edges, $L_{u,v} \neq R_{u,v}$. This will be always assumed in the proofs below without specifically recalling the reason again. We classify arcs according to the colors of their incident faces. Let $E_{i,j}$ be the set of arcs (u, v) in $\boldsymbol{G}$ such that $\Lambda(L_{u,v}) = i$ and $\Lambda(R_{u,v}) = j$. It is easy to see that each arc is exactly in one such set. Hence, the following lemma is evident and can be given without proof.

Lemma 1. *For any face λ-coloring of a planar geometric graph G,*

$$\sum_{i=1}^{\lambda} \sum_{j=1, j\neq i}^{\lambda} |E_{i,j}| = 2|E|.$$

For any of $\lambda(\lambda-1)$ ordered pairs of two distinct colors a and b in the coloring Λ, we define the digraph $D(G;a,b)$ as follows: The vertex set of the digraph D is V and the arc set of D is

$$\bigcup_{i\in[1,\lambda]\setminus\{b\}, j\in[1,\lambda]\setminus\{a\}} E_{i,j}.$$

Along with this definition, for $i \neq b$, $j \neq a$, and $i \neq j$, we say that $E_{i,j}$ is in $D(G;a,b)$. Next consider the following characteristic function

$$\chi_{a,b}(E_{i,j}) = \begin{cases} 1 \text{ if } E_{i,j} \text{ is in } D(G;a,b), \text{ and} \\ 0 \text{ otherwise.} \end{cases}$$

We claim that every set $E_{i,j}$ is in exactly $\lambda^2-3\lambda+3$ different digraphs $D(G;a,b)$ for some $a \neq b$.

Lemma 2. *For any face λ-coloring of a planar geometric graph G,*

$$\sum_{a=1}^{\lambda} \sum_{b=1, b\neq a}^{\lambda} \chi_{a,b}(E_{i,j}) = \lambda^2 - 3\lambda + 3.$$

Proof. Let $i, j \in [1,\lambda]$, $i \neq j$ be fixed. For any two distinct colors a and b of the λ-coloring of G, $\chi_{a,b}(E_{i,j}) = 1$ only if either $i = a$, or $j = b$, or i and j are different from a and b. There are $(\lambda-1)+(\lambda-2)+(\lambda-2)(\lambda-3)$ such colorings. The lemma follows by simple counting. □

The following lemma gives a key property of the digraph $D(G;a,b)$.

Lemma 3. *Given a face λ-coloring of a planar geometric graph G with no cut edges, and the corresponding digraph $D(G;a,b)$. Every face of $D(G;a,b)$, which has color a, constitutes a counter clockwise directed cycle, and every face which has color b, constitutes a clockwise directed cycle. All arcs on such cycles are unidirectional. Moreover, each arc of $D(G;a,b)$ incident to faces having colors different from either a or b is bidirectional.*

Proof. Let G be a planar geometric graph with a face λ-coloring Λ with colors a, b and $\lambda-2$ other colors. Consider $D(G;a,b)$. The sets $E_{a,x}$ are in $D(G;a,b)$ for each color $x \neq a$. Let f be a face and let $\{u,v\}$ be an edge of f so that $L_{u,v} = f$. Let f' be the other face incident to $\{u,v\}$; hence $R_{u,v} = f'$.

Since G has no cut edges, $f \neq f'$, and since $\Lambda(f') \neq a$, the arc $(u,v) \in \bigcup_{x\neq a} E_{a,x}$ and hence the arc (u,v) is in $D(G;a,b)$. Since $\{u,v\}$ was an arbitrary edge of f, f will induce a counter clockwise cycle in $D(G;a,b)$ (See Figure 1.)

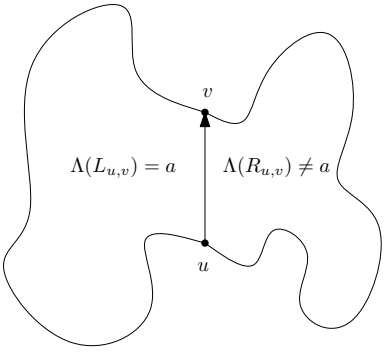


Fig. 1. (u, v) is in $D(G; a, b)$ if $\Lambda(L_{u,v}) = a$ and therefore the edges in the face $L_{u,v}$ form a counter clockwise directed cycle in $D(G; a, b)$

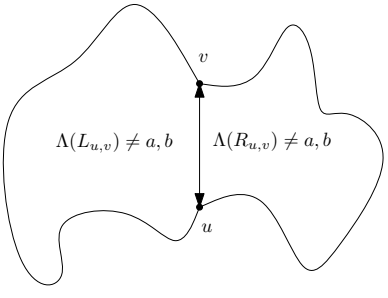


Fig. 2. A bidirectional arc is in $D(G; a, b)$ if its incident faces have color different than a and b

The fact that every face which has color b induces a clockwise cycle in $D(G; a, b)$ is similar.

Finally consider an edge $\{u, v\}$ such that $c = \Lambda(L_{u,v}) \neq a, b$ and $d = \Lambda(R_{u,v}) \neq a, b$ (See Figure 2.) Hence $(u, v) \in E_{c,d}$ which is in $D(G; a, b)$ and similarly $(v, u) \in E_{d,c}$ which is also in $D(G; a, b)$. This proves the lemma. □

We are ready to prove Theorem 1.

Proof (Theorem 1). Let G be a planar geometric graph having no cut edges. Let Λ be a face λ-coloring of G with colors a, b, and other $\lambda - 2$ colors. Suppose colors a and b are such that the corresponding digraph $D(G; a, b)$ has the minimum number of arcs. Consider $\overline{A}$ the average number of arcs in all digraphs arising from Λ. Thus,

$$\overline{A} = \frac{1}{\lambda(\lambda - 1)} \sum_{a=1}^{\lambda} \sum_{b=1, b \neq a}^{\lambda} |D(G; a, b)|, \text{ where}$$

$$|D(G; a, b)| = \sum_{i=1}^{\lambda} \sum_{j=1, j \neq i}^{\lambda} \chi_{a,b}(E_{i,j}) |E_{i,j}|.$$

By Lemma 1 and Lemma 2,

$$\begin{aligned}\overline{A} &= \frac{1}{\lambda(\lambda-1)} \sum_{a=1}^{\lambda} \sum_{b=1,b\neq a}^{\lambda} \sum_{i=1}^{\lambda} \sum_{j=1,j\neq i}^{\lambda} \chi_{a,b}(E_{i,j})|E_{i,j}| \\ &= \frac{1}{\lambda(\lambda-1)} \sum_{i=1}^{\lambda} \sum_{j=1,j\neq i}^{\lambda} (\lambda^2-3\lambda+3)|E_{i,j}| \\ &= \frac{2(\lambda^2-3\lambda+3)}{\lambda(\lambda-1)}|E| \\ &= \left(2-\frac{4\lambda-6}{\lambda(\lambda-1)}\right)|E|.\end{aligned}$$

Hence $D(G;a,b)$ has at most the desired number of arcs.

To prove the strong connectivity of $D(G;a,b)$, consider any path, say $u = u_0, u_1, \ldots, u_n = v$, in the graph G from u to v. We prove that there exists a directed path from u to v in $D(G;a,b)$. It is enough to prove that for all i there is always a directed path from u_i to u_{i+1} for any edge $\{u_i, u_{i+1}\}$ of the above path. We distinguish several cases.

- Case 1. $\Lambda(L_{u_i,u_{i+1}}) = a$. Then $(u_i, u_{i+1}) \in E_{a,\omega}$ where $\omega = \Lambda(R_{u_i,u_{i+1}})$. Since $E_{a,\omega}$ is in $D(G;a,b)$, the arc (u_i, u_{i+1}) is in $D(G;a,b)$. Moreover, the stretch factor of $\{u_i, u_{i+1}\}$ is one.
- Case 2. $\Lambda(L_{u_i,u_{i+1}}) = b$. Hence, (u_i, u_{i+1}) is not in $D(G;a,b)$. However, by Lemma 3, the face $L_{u_i,u_{i+1}} = R_{u_{i+1},u_i}$ constitutes a clockwise directed cycle, and therefore, a directed path from u_i to u_{i+1}. It is easy to see that the stretch factor of $\{u_i, u_{i+1}\}$ is not more than the size of the face $L_{u_i,u_{i+1}}$ minus one, which is at most $\Phi(G)-1$.
- Case 3. $\Lambda(L_{u_i,u_{i+1}}) \neq a, b$. Suppose $\Lambda(L_{u_i,u_{i+1}}) = c$. Three cases can occur.
  - $\Lambda(R_{u_i,u_{i+1}}) = a$. Hence, (u_i, u_{i+1}) is not in $D(G;a,b)$. However, by Lemma 3, there exists a counter clockwise directed cycle around face $R_{u_i,u_{i+1}} = L_{u_{i+1},u_i}$, and consequently a directed path from u_i to u_{i+1}. The stretch factor is at most the size of face $R_{u_i,u_{i+1}}$ minus one, which is at most $\Phi(G)-1$.
  - $\Lambda(R_{u_i,u_{i+1}}) = b$. By Lemma 3, there exists a clockwise directed cycle around face $R_{u_i,u_{i+1}}$. This cycle contains (u_i, u_{i+1}), and in addition the stretch factor of $\{u_i, u_{i+1}\}$ is one.
  - $\Lambda(R_{u_i,u_{i+1}}) = d \neq a, b, c$. By the construction, $D(G;a,b)$ has both arcs (u_i, u_{i+1}) and (u_{i+1}, u_i). Again, the stretch factor of $\{u_i, u_{i+1}\}$ is one.

This proves the theorem. □

As indicated in Theorem 1 the number of arcs in the orientation depends on the number λ of colors. Thus, for specific values of λ we have the following table of values:

λ	3	4	5	6	7
$2-\frac{4\lambda-6}{\lambda(\lambda-1)}$	1	$\frac{7}{6}$	$\frac{13}{10}$	$\frac{7}{5}$	$\frac{31}{21}$

Regarding the complexity of the algorithm, this depends on the number λ of colors being used. For example, computing a 4-coloring can be done in $O(n^2)$ [12]. Finding the digraph with minimum number of arcs among the twelve possible digraphs can be done in linear time. Therefore, for $\lambda = 4$, the orientation can be computed in $O(n^2)$. For $\lambda = 5$ a 5-coloring can be found in $O(n)$ time. For geometric planar subgraphs of unit disk graphs and location aware nodes there is a local 7-coloring (see [4]). For more information on colorings the reader is advised to look at [8]. We also have the following corollary.

Corollary 1. *Let $G = (V, E, F)$ be a geometric planar triangulation. There exists a strongly connected orientation $\boldsymbol{G}$ with at most $7(|V| - 2)/2$ arcs and stretch factor of 2.*

3 Orientations with $|E|$ Arcs

Theorem 1 shows that every geometric planar graph G without cut edges has a strong orientation with bounded stretch factor and at most $\left(2-\frac{4\lambda-6}{\lambda(\lambda-1)}\right)\cdot|E|$ arcs. In this section we show that one can orient every edge in exactly one direction only and still obtain a strong orientation. However the stretch factor will increase.

Consider a geometric planar graph $G(V, E, F)$ having no cut edges and a face λ-coloring Λ of G with colors $[1, \lambda]$. Let $\boldsymbol{G}$ be the orientation assigning two opposite directions to each edge of E. Recall that $L_{u,v} \neq R_{u,v}$ since G does not have cut edges and $E_{i,j}$ is the set of arcs (u, v) in $\boldsymbol{G}$ such that $\Lambda(L_{u,v}) = i$ and $\Lambda(R_{u,v}) = j$. Clearly, these sets are pairwise disjoint. We define the digraph $D(G; \Lambda)$ as follows: The vertex set of the digraph $D(G; \Lambda)$ is V and the arc set of $D(G; \Lambda)$ is $\bigcup_{i<j\leq\lambda} E_{i,j}$.

It is not difficult to observe that in $D(G; \Lambda)$ exactly one direction is assigned to every edge of G.

Theorem 2. *Let $G(V, E, F)$ be a geometric planar graph which is 2-edge connected. For any face λ-coloring Λ of G, the digraph $D(G; \Lambda)$ is strongly connected, has exactly $|E|$ arcs, and its stretch factor is at most $(\Phi(G) - 1)^{\lceil\frac{\lambda+1}{2}\rceil}$.*

Proof. We already observed above that $D(G; \Lambda)$ has $|E|$ arcs. We prove the following two statements.

1. We first prove by induction on k that if $\{u, v\} \in E$ so that $\Lambda(L_{u,v}) = k$ then if (u, v) is in $D(G; \Lambda)$ then there is also a directed path from v to u in $D(G; \Lambda)$ of length at most $(\Phi(G) - 1)^k$ such that every arc on this path is incident to a face of color at most k.
2. Second we prove that for every k if $\{u, v\} \in E$ so that $\Lambda(R_{u,v}) = k$ then if (v, u) is in $D(G; \Lambda)$ then there is also a directed path from u to v in $D(G; \Lambda)$ of length at most $(\Phi(G) - 1)^{\lambda-k+1}$ such that every arc on this path is incident to a face of color at least k.

The theorem follows easily. Indeed, let $\{u, v\} \in E$ so that $\Lambda(L_{u,v}) < \Lambda(R_{u,v})$. The arc (u, v) constitutes the required directed path from u to v. We exhibit required directed path from v to u as follows. If $\Lambda(L_{u,v}) \leq \lceil \frac{\lambda}{2} \rceil$, then the directed path from v to u exists by first statement. If $\Lambda(L_{u,v}) > \lceil \frac{\lambda}{2} \rceil$, then since $L_{u,v} = R_{v,u}$, $\Lambda(R_{v,u}) > \lceil \frac{\lambda}{2} \rceil$ and since $(u, v) \in E$, the required directed path from v to u exists by the second statement above.

Next we give the proof of the statement 1 above. Base step is $\Lambda(L_{u,v}) = 1$. Hence, the other face incident to $\{u, v\}$ has color $j > 1$. Therefore, $(u, v) \in E_{1,j}$ which is in $D(G; \Lambda)$ by definition. We have the same conclusion for any other arc of the face $L_{u,v}$ and hence this face will induce a directed cycle in $D(G; \Lambda)$, which provides a desired directed path from v to u. The length of this path is obviously at most $\Phi(G) - 1$. Also every arc of this path is obviously incident to a face of color 1. This proves the base case.

In the inductive step we assume the statement is true for all $l \leq k-1$. We prove it for k. Assume $\Lambda(L_{u,v}) = k$. If $k < \Lambda(R_{u,v}) = k'$, then $(u, v) \in E_{k,k'}$ which is in $D(G; \Lambda)$ by definition. To construct a directed path from v to u, consider the face $L_{u,v}$ and any edge $\{a, b\}$ incident to this face so that $L_{a,b} = L_{u,v}$. If $\Lambda(L_{a,b}) < \Lambda(R_{a,b})$, the arc (a, b) is in $D(G; \Lambda)$, and $\Lambda(L_{a,b}) = k \leq k$ as required. Otherwise the arc (b, a) is in $D(G; \Lambda)$ and also $l = \Lambda(L_{b,a}) = \Lambda(R_{a,b}) < \Lambda(L_{a,b}) = k$ (See Figure 3.) Thus, by inductive hypothesis, there is a directed path from a to b in $D(G; \Lambda)$ of length at most $(\Phi(G)-1)^l$ such that every arc of this path is incident to a face of color at most $\Lambda(L_{b,a}) = l < k$. At most $(\Phi(G) - 1)$ arcs of $L_{u,v}$ will be replaced in this way, so the length of the desired path from v to u is at most $(\Phi(G) - 1)^k$. If $k > \Lambda(R_{u,v}) = k'$, then (u, v) is not in $D(G; \Lambda)$.

Finally we give the proof of the statement 2 above. Base step is $\Lambda(R_{u,v}) = \lambda$ which is trivially true since (v, u) is not in $D(G; \Lambda)$. In the inductive step we assume the statement is true for all $l \geq \lambda - k + 1$. We prove it for $\lambda - k$. Assume $\Lambda(R_{u,v}) = \lambda - k$. If $\lambda - k > \Lambda(L_{u,v}) = k'$, then $(u, v) \in E_{k',\lambda-k}$ which is in $D(G; \Lambda)$ by definition and hence (v, u) is not in $D(G; \lambda)$. Hence suppose $\lambda - k < \Lambda(L_{u,v}) = k'$. Hence (v, u) is in $D(G; \Lambda)$. To construct a directed path from u to v, consider the face $L_{u,v}$ and any edge $\{a, b\}$ incident to this face so that $R_{a,b} = L_{u,v}$. If $\Lambda(L_{a,b}) < \Lambda(R_{a,b})$, the arc (a, b) is in $D(G; \Lambda)$, and $\Lambda(R_{a,b}) = k' \geq \lambda - k$ as required. Otherwise the arc (b, a) is in $D(G; \Lambda)$ and also

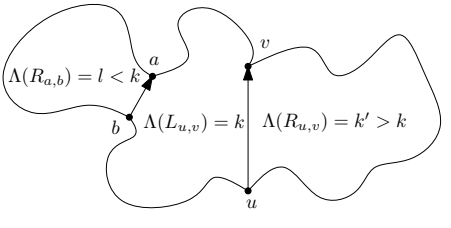


Fig. 3. The figure shows how to find a directed path from v to u when (u, v) is in $D(G; \Lambda)$. If $\{a, b\}$ is any edge incident to $L_{u,v}$ then by inductive hypothesis there is a path from a to b of length at most $(\Phi - 1)^l$.

$\Lambda(R_{a,b}) = k'$. Thus, by inductive hypothesis, there is a directed path from a to b in $D(G;\Lambda)$ of length at most $(\Phi(G)-1)^{\lambda-k'+1}$ such that every arc of this path is incident to a face of color at least $\Lambda(R_{a,b}) = k' > \lambda - k$. At most $(\Phi(G)-1)$ arcs of $L_{u,v}$ will be replaced in this way, so the length of the desired path from u to v is at most $(\Phi(G)-1)^{\lambda-k'+1+1} \leq (\Phi(G)-1)^{k+1}$, since $k' \geq \lambda - k + 1$. This proves the theorem. □

As a corollary we obtain the following result on triangulations.

Corollary 2. *Let $G(V,E,F)$ be a geometric planar triangulation, and let Λ be its face 4-coloring. The digraph $D(G;\Lambda)$ is strongly connected, has exactly $|E|$ arcs, and its stretch factor is at most 8.*

4 Orientations with Less than $|E|$ Arcs

By considering more sophisticated face colorings, we can decrease the number of arcs below $|E|$ in a strong orientation and still maintain a bounded stretch factor. Define

$$D'(G;\Lambda) = D(G;\Lambda) - E_{\lceil \frac{\lambda-1}{2} \rceil, \lceil \frac{\lambda+1}{2} \rceil}.$$

We use the following result about acyclic coloring of planar graphs. A (proper vertex) coloring is acyclic if every subgraph induced by any two colors is acyclic.

Theorem 3. *[2] Every planar graph has an acyclic coloring with 5 colors.*

Lemma 4. *Let $T = (V,E)$ be a forest and $k \geq 1$ an integer. There exists a set of vertices $S \subseteq V$ such that the subgraph of T induced by S is a forest of trees of diameter at most $4k$ and has at least $\frac{k}{k+1}|E|$ arcs.*

Proof. In this proof all indices will be considered modulo $2k+2$. Root every component of T at any vertex and consider the partition of V into k sets $V_0, V_1, \ldots, V_{2k+1}$ such that the set

$$V_\ell = \{x \in V : \text{ distance of } x \text{ from the root of its component is } \ell \bmod 2k+2\}.$$

Now consider the following k forests of trees of diameter at most $4k$. For $m = 0, 1, \ldots, k-1$, let $G^m = (V^m, E^m)$ where

$$V^m = V_{0-2m} \cup V_{1-2m} \cup \cdots \cup V_{2k-2m}\,,$$
$$E^m = \{\{x,y\} \in E : x \in \cup_{i=0-2m}^{2k-1-2m} V_i,\ y \in \cup_{i=1-2m}^{2k-2m} V_i\}.$$

It is not difficult to see that every G^m is, in fact, an induced subgraph of T. If one of the graphs G^m has at least $\frac{k}{k+1}|E|$ arcs, we are done. On the other hand, only edges of T that are not included in G^m for given m are edges $\{x,y\}$ such that $x \in V_{2k-2m}$ and $y \in V_{2k+1-2m}$ and edges $\{x,y\}$ such that $x \in V_{2k+1-2m}$ and $y \in V_{2k+2-2m}$, i.e. edges of stars centered at vertices in $V_{2k+1-2m}$. Since

G^m has less than $\frac{k}{k+1}|E|$ edges, there is at least $\frac{1}{k+1}|E|$ such edges. This must be true for all $m = 0, 1, \ldots, k-1$, and these edge sets are pairwise disjoint for distinct values of m. Hence the graph $G^k = (V^k, E^k)$ such that

$$V^k = V_2 \cup V_3 \cup \cdots \cup V_{2k+2},$$
$$E^k = \{\{x, y\} \in E : x \in \cup_{i=2}^{2k+1} V_i, y \in \cup_{i=3}^{2k+2} V_i\}$$

is the forest of trees of diameter at most $4k$, is induced in T and has at least $\frac{k}{k+1}|E|$ edges. □

The main theorem is as follows.

Theorem 4. *Let $G(V, E, F)$ be a geometric planar graph which is 3-edge connected, and let $k \geq 1$ be an integer. There exists a face 6-coloring Λ of G so that the digraph $D'(G; \Lambda)$ is strongly connected, has at most $(1 - \frac{k}{10(k+1)})|E|$ arcs, and its stretch factor is at most $\Phi^2(G)(\Phi(G) - 1)^{2k+4}$.*

Proof. Let $G^\star$ be the dual graph of G. Since G is 3-edge connected $G^\star$ is a simple graph and every edge of G is crossed by a unique edge of $G^\star$. Consider an acyclic 5-coloring of $G^\star$ which exists by Theorem 3. Among all ten pairs of colors in the 5-coloring choose a pair so that the forest H induced by vertices colored with these two colors has at least $\frac{|E|}{10}$ edges. By Lemma 4, in this forest we can select a set of induced trees each of diameter at most $4k$ such that they will together span at least $\frac{k}{k+1}\frac{|E|}{10}$ edges of $G^\star$.

We are now ready to color faces of G and define a face 6-coloring Λ of G as follows: We use colors 3 and 4 to color faces corresponding to vertices of trees selected in the dual $G^\star$, and we use colors 1,2,5, and 6 to properly color remaining faces of G.

With $\lambda = 6$, we let color $\alpha = \lceil \frac{\lambda-1}{2} \rceil = 3$ and $\beta = \lceil \frac{\lambda+1}{2} \rceil = 4$. By our construction, the pair $\{\alpha, \beta\}$ must appear at least $\frac{k}{k+1}\frac{|E|}{10}$ times in the face 6-coloring of G. This gives the required bound on the number of arcs of the graph $D'(G; \Lambda)$.

Statements 1 and 2 given in the proof of Theorem 2 imply that if $\{u, v\} \in E$ such that $\Lambda(L_{u,v}) < \Lambda(R_{u,v})$ and if either $\Lambda(L_{u,v}) \neq \alpha$ or $\Lambda(R_{u,v}) \neq \beta$, then $D'(G; \Lambda)$ contains directed path from u to v as well as from v to u. Indeed, if $\Lambda(L_{u,v}) \geq \alpha$, then $\Lambda(R_{u,v}) > \beta$, and we can apply statement 2. Similarly if $\Lambda(R_{u,v}) \leq \beta$, then $\Lambda(L_{u,v}) < \alpha$, and we can apply statement 1. Obviously the pair of colors α and β is not incident to any arc on these paths, so these paths exist in $D'(G; \Lambda)$. Note that these paths have bounded stretch factor as in Theorem 2, in particular $(\Phi(G) - 1)^4$.

To complete the proof we consider $\{u, v\} \in E$ such that $\Lambda(L_{u,v}) < \Lambda(R_{u,v})$ and $\Lambda(L_{u,v}) = \alpha$ and $\Lambda(R_{u,v}) = \beta$. Hence these edges do not occur in $D'(G; \Lambda)$. The edge $\{L_{u,v}, R_{u,v}\}$ of the dual $G^\star$ belongs to one of the selected trees, say T, of diameter at most $4k$. The vertices of this tree correspond to faces of G that are colored with color 3 or 4. Since G is 3-edge connected, there is a path

P from u to v in G along these faces such that for each edge of P one of its incident faces has color different from 3 and 4. Hence for each edge of P the digraph $D'(G;\Lambda)$ contains directed paths (in both directions) of length at most $(\Phi(G)-1)^4$. Since the maximum degree of T is $\Phi(G)$ and the diameter of T is at most $4k$, T has at most $\Phi(G)(\Phi(G)-1)^{2k}$ vertices. Each of these vertices corresponds to a face of degree at most $\Phi(G)$. Hence the length of P is at most $\Phi^2(G)(\Phi(G)-1)^{2k}$. Finally we conclude that $D'(G;\Lambda)$ contains a directed path from u to v and from v to u both of length at most $\Phi^2(G)(\Phi(G)-1)^{2k+4}$.

It follows that $D'(G;\Lambda)$ is strongly connected and has stretch factor at most $\Phi^2(G)(\Phi(G)-1)^{2k+4}$. □

Note that with a more careful counting argument the bound on the stretch factor in Theorem 4 can be improved by at least half. Using the following theorem we can further decrease the number of arcs in a strong orientation of G and still keep the stretch factor bounded.

Theorem 5. *Let $G=(V,E,F)$ be a 3-connected planar graph. Then G contains a spanning 2-edge connected subgraph G' with at most $|E|-\lfloor\frac{|E|+3}{3\Phi(G)}\rfloor$ edges and $\Phi(G')\leq 2\Phi(G)$.*

Proof. Since G is 3-connected, the dual $G^\star$ is a simple graph. Let T be a spanning tree of $G^\star$. Obviously, the maximum degree of T is at most $\Phi(G)$ and T has least $\frac{|E|}{3}+2$ vertices. The later follows from Euler's formula and the fact that minimum degree of G is 3. Moreover, T has a matching of size at least $\lfloor\frac{|E|+3}{3\Phi(G)}\rfloor$. Indeed, one can obtain such a matching M by recursively adding a pendant edge (an edge adjacent to a leaf) of remaining components of T into M and then removing all remaining edges incident to this edge. Each such operation adds one edge into M and removes at most $\Phi(G)$ edges (including the edge itself) from T. Since T has $\frac{|E|}{3}+1$ edges at the beginning, the bound follows.

To obtain G', we merge corresponding faces in G for every edge in M. Since M is a matching $\Phi(G')\leq 2\Phi(G)$ and G' will be 2-connected. Obviously G' has required number of edges. □

5 Conclusion

We presented algorithms for directing edges of a planar graph having no cut edges such that the resulting digraph is strongly connected and has bounded stretch factor which depends solely on the size of the faces of the original planar graph. An interesting question arises how to construct planar graphs having no cut edges. Although it is well-known how to construct such planar spanners starting from a set of points (e.g., Delaunay triangulation) there are no known constructions in the literature of "local" spanners from UDGs which also guarantee planarity and 2-edge connectivity at the same time.

References

[1] Bhattacharya, B., Hu, Y., Kranakis, E., Krizanc, D., Shi, Q.: Sensor network connectivity with multiple directional antennae of a given angular sum. In: IEEE IPDPS, 23rd International Parallel and Distributed Processing Symposium, May 25-29, pp. 344–351 (2009)

[2] Borodin, O.V.: On acyclic colorings of planar graphs. Discrete Math. 25(3), 211–236 (1979)

[3] Caragiannis, I., Kaklamanis, C., Kranakis, E., Krizanc, D., Wiese, A.: Communication in wireless networks with directional antennas. In: SPAA 2008: Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures, pp. 344–351. ACM Press, New York (2008)

[4] Czyzowicz, J., Dobrev, S., Gonzalez-Aguilar, H., Kralovic, R., Kranakis, E., Opatrny, J., Stacho, L., Urrutia, J.: Local 7-coloring for planar subgraphs of unit disk graphs. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 170–181. Springer, Heidelberg (2008)

[5] Dobrev, S., Kranakis, E., Krizanc, D., Morales, O., Opatrny, J., Stacho, L.: Strong connectivity in sensor networks with given number of directional antennae of bounded angle (2009) (to appear)

[6] Fukunaga, T.: Graph orientations with set connectivity requirements. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 265–274. Springer, Heidelberg (2009)

[7] Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Transactions on Information Theory 46(2), 388–404 (2000)

[8] Jensen, T., Toft, B.: Graph coloring problems. Wiley-Interscience, New York (1996)

[9] Kranakis, E., Krizanc, D., Williams, E.: Directional versus omnidirectional antennas for energy consumption and k-connectivity of networks of sensors. In: Higashino, T. (ed.) OPODIS 2004. LNCS, vol. 3544, pp. 357–368. Springer, Heidelberg (2005)

[10] Nash-Williams, C.S.J.A.: On orientations, connectivity and odd vertex pairings in finite graphs. Canad. J. Math. 12, 555–567 (1960)

[11] Parker, R., Rardin, R.: Guaranteed performance heuristics for the bottleneck traveling salesman problem. Oper. Res. Lett. 2(6), 269–272 (1984)

[12] Robertson, N., Sanders, D., Seymour, P., Thomas, R.: The four-colour theorem. J. Comb. Theory Ser. B 70(1), 2–44 (1997)

[13] Yi, S., Pei, Y., Kalyanaraman, S., Azimi-Sadjadi, B.: How is the capacity of ad hoc networks improved with directional antennas? Wireless Networks 13(5), 635–648 (2007)

[14] Zhang, H., He, X.: On even triangulations of 2-connected embedded graphs. SIAM J. Comput. 34(3), 683–696 (2005)

A Linear Time Algorithm for the Minimum Spanning Caterpillar Problem for Bounded Treewidth Graphs

Michael J. Dinneen and Masoud Khosravani*

Department of Computer Science
The University of Auckland
Auckland, New Zealand
{mjd,masoud}@cs.auckland.ac.nz

Abstract. We consider the Minimum Spanning Caterpillar Problem (MSCP) in a graph where each edge has two costs, spine (path) cost and leaf cost, depending on whether it is used as a spine or a leaf edge. The goal is to find a spanning caterpillar in which the sum of its edge costs is the minimum. We show that the problem has a linear time algorithm when a tree decomposition of the graph is given as part of the input. Despite the fast growing constant factor of the time complexity of our algorithm, it is still practical and efficient for some classes of graphs, such as outerplanar, series-parallel (K_4 minor-free), and Halin graphs. We also briefly explain how one can modify our algorithm to solve the Minimum Spanning Ring Star and the Dual Cost Minimum Spanning Tree Problems.

Keywords: spanning caterpillars, treewidth, networks topology, optimization.

1 Introduction

By a caterpillar we mean a tree that reduces to a path by deleting all its leaves. We refer to the remaining path as the *spine* of the caterpillar. The edges of a caterpillar H can be partitioned to two sets, the spine edges, $\mathcal{B}(H)$, and the leaf edges, $\mathcal{L}(H)$. Let $G = (V, E)$ be a graph. Also let $b : E \to \mathbb{N}$ and $l : E \to \mathbb{N}$ be two (cost) functions. For each caterpillar H as a subgraph of G we define the cost of H by

$$c(H) := \sum_{e \in \mathcal{B}(H)} b(e) + \sum_{e' \in \mathcal{L}(H)} l(e').$$

In the Minimum Spanning Caterpillar Problem (MSCP) [10] one wants to find a caterpillar with the minimum cost that contains all vertices. The MSCP is $\mathcal{NP}$-complete for general graphs [11]. In this paper we consider the problem when

* Research supported by the Computer Science Department Doctoral Scholarship, University of Auckland.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 237–246, 2010.

the input is restricted to bounded treewidth graphs. That is, we assume that a tree decomposition with a fixed width of the graph is given as part of the input.

One application of this problem is to find a cost effective subnetwork within a large network. For example, one may want to find a linearly connected backbone to place routers with computers attached. Here each cost function represents a different technology that is used to connect vertices on the backbone (that consists of routers or hubs) and the leaves (i.e. computers) to the backbone. As another application of the MSCP, Tan and Zhang [11] used it to solve some problems concerning the Consecutive Ones Property problem.

It is well known that graph structures that are expressible by Monadic Second Order Logic (MSOL) [4] are recognizable in linear time on bounded treewidth graphs. The same is true for those optimization problems that can be defined by Extended Monadic Second Order Logic (EMSOL); see the survey of Hlineny et. al. [7]. The main disadvantage of these methods is that they are very hard to implement, even for small values of k, where k is the treewidth of a graph. So here we present a simple linear time algorithm for finding an optimal caterpillar in a graph with bounded treewidth. Although the hidden constant factor in the asymptotic notation of the algorithm grows very fast, but for some graphs that appears in applications it is still practical, like outerplanar, series-parallel (K_4 minor-free), and Halin graphs.

Also we explain briefly how one can tackle two other related $\mathcal{NP}$-hard problems: the Minimum Spanning Ring Star Problem (MSRSP) [1,8] where the goal is to find a minimum spanning subgraph (star ring) that consists of a cycle and vertices of degree one that are connected to it, and the Dual Cost Minimum Spanning Tree Problem (DCMSP), where the cost of an edge incident to a leaf is different from the other edges. As far as we know it is the first time that these problems are studied from this point of view.

In the next section we present the required definitions formally. In particular, we introduce the k-parse data structure as an alternative to the smooth tree decomposition. In Section 3 we give a dynamic programming algorithm that solves the MSCP in linear time for graphs that their tree decompositions are also given as inputs. The proof of correctness of the algorithm is presented in Section 4. The paper ends with a conclusion and some open problems.

2 Preliminaries

In this paper we suppose that all graphs are undirected and they have no multiple edges or loops.

A tree (path) decomposition [9] of a graph $G = (V, E)$ is a pair $(\{X_i, i \in I\}, T = (I, F))$, with $\{X_i, i \in I\}$ a collection of subsets of V, that are called *bags*, and $T = (I, F)$ a tree (path), such that

1. $\bigcup_{i \in I} X_i = V$.
2. For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.
3. For all $v \in V$, $T_v = \{i \in I \mid v \in X_i\}$ forms a connected subtree of T.

The width of a tree (path) decomposition $(\{X_i, i \in I\}, T = (I, F))$ is defined as $\max_{i \in I} |X_i| - 1$. The treewidth (pathwidth) of G, $tw(G)$ ($pw(G)$), is the minimum width over all tree (path) decompositions of G. A tree (path) decomposition of width k is *smooth* if each bag contains $k+1$ vertices and adjacent bags differ by exactly one vertex. For a survey on treewidth refer to Bodlaender [3].

In the rest of this section we introduce k-parse data structure as a linear and detailed representation of a smooth tree decomposition. It enables us to follow explicitly the process of adding edges that is not clearly expressed in a smooth tree decomposition. That makes the presentation easier and the implementation more straightforward.

A $(k+1)$-*boundaried graph* [6] is a pair (G, ∂) of a graph $G = (V, E)$ and an injective function ∂ from $\{0, \ldots, k\}$ to V. The image of ∂ is the set of *boundaried vertices* and is denoted by $Im(\partial)$. When it is clear from the context, we abuse the notation and refer to $Im(\partial)$ as ∂.

Given a path decomposition of width k of a graph, one can represent the graph by using strings of (unary) operators from the following *operator set* $\Sigma_k = V_k \cup E_k$ (see [5]):

$$V_k = \{\textcircled{0}, \ldots, \textcircled{k}\} \quad \text{and} \quad E_k = \{\boxed{i\ j} \mid 0 \leq i < j \leq k\}.$$

Where V_k is the set of vertex operators and E_k is the set if edge operators.

To generate a graph from a smooth tree decomposition of width k, an additional (binary) operator $\oplus$, called *circle plus*, is added to Σ_k. The semantics of these operators on $(k+1)$-boundaried graphs G and H of are as follows:

- $G\,\textcircled{i}$ Add an isolated vertex to the graph G, and label it as the new boundary vertex i.
- $G\,\boxed{i\ j}$ Add an edge between boundaried vertices i and j of G (ignore if operation causes a multi-edge).
- $G \oplus H$ Take the disjoint union of G and H except that equal-labeled boundary vertices of G and H are identified.

It is syntactically incorrect to use the operator $\boxed{i\ j}$ without being preceded by both $\textcircled{i}$ and $\textcircled{j}$, and the operator $\oplus$ must be applied to graphs with the same sized boundaries. A graph described by a string (tree, if $\oplus$ is used) of these operators is called a k-*parse*, and has an implicit labeled boundary ∂ of at most $k+1$ vertices. By convention, a k-parse begins with the axiom operator string $[\textcircled{0}, \textcircled{1}, \ldots, \textcircled{k}]$ which represents the edgeless graph of order $k+1$. Throughout this paper, we refer to a k-parse and the graph it represents interchangeably.

Let $G = (g_0, g_1, \ldots, g_n)$ be a k-parse and $Z = (z_0, z_1, \ldots, z_m)$ be any sequence of operators over Σ_k. The *concatenation* $(\cdot)$ of G and Z is defined as

$$G \cdot Z = (g_0, g_1, \ldots, g_n, z_0, z_1, \ldots, z_m).$$

(For the treewidth case, G and Z are viewed as two connected subtree factors of a parse tree $G \cdot Z$ instead of two parts of a sequence of operators.)

Bodlaender in [2] presented a linear time algorithm that makes a smooth tree decomposition from any tree decomposition of a graph and for each smooth tree decomposition one can easily construct a k-parse representation.

3 Finding a Minimum Spanning Caterpillar in a Bounded Treewidth Graph

In this section we show that the problem of finding a minimum spanning caterpillar in a bounded treewidth graph with n vertices has an algorithm that runs in linear time, assuming the tree decomposition is given alongside the input graph G. Throughout this section we suppose that the treewidth of G is k and G is represented as a k-parse $G = (g_0, \ldots, g_m)$.

The main idea is to use a forest of at most $k+1$ different caterpillars as a partial solution, each has at least one vertex in the boundary set $\partial = \{0, \ldots, k\}$. To this end we *code* the information of each partial solution in a state vector $S = (A, B)$. Where A is a $(k+1)$-tuple $(a_0, \ldots, a_k)$. Each a_i represents a label for the boundary vertex i from the set $\{H, S, C, I, L\}$. The labels H, S, C, I, and L are characteristics of the boundary vertices in a partial solution. They stand for *head, spine, center* (of a star), *isolated vertex*, and *leaf*, respectively. The set B is a partition set of ∂. If any two boundary vertices belong to the same element of B, then they belong to the same connected component of a partial solution that is represented by B. See Figure 1.

In accordance with our dynamic programming approach, we use a table $\mathcal{T}$ that has rows indexed by state vectors and columns indexed by k-parse operators from g_k to g_m. Each entry in a row (A, B) and a column g_i in $\mathcal{T}$ is a pair (X, x). Where $X \in \{true, false\}$ shows if the entry represents a valid forest of caterpillars and x is the minimum total cost among the partial solutions represented by the state vector (A, B) at column g_i.

We initialize all entries of $\mathcal{T}$ to the value $(false, \infty)$. Due to our convention for the first $k+1$ operators we have $g_i = (i)$, $0 \le i \le k$. So at the first step we assign the value $(true, 0)$ to $\mathcal{T}((A, B), g_k)$, where $A = (I, \ldots, I)$ and $B = \{\{0\}, \{1\}, \ldots, \{k\}\}$. Suppose we have computed the entries of $\mathcal{T}$ up to the

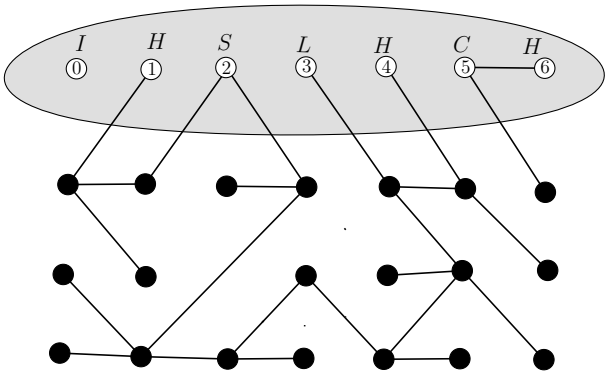


Fig. 1. A forest of caterpillars as a partial solution, $A = (I, H, S, L, H, C, H)$ and $B = \{\{0\}, \{1, 2, 3, 4\}, \{5, 6\}\}$

$(p-1)$th column (which is indexed by g_{p-1} operation). Then we scan the column $p-1$ to find entries that their first coordinates are *true*. By considering each *true* entry in column $p-1$, we update the entries in column p by the following rules.

Vertex operator $\textcircled{i}$*:* Suppose that g_p is a vertex operator that introduces a new vertex as the boundary vertex i. Then for each $\mathcal{T}((A,B),g_{p-1}) = (true,x)$ let $B_i \subseteq \partial$ be the element of B that contains boundary value i. If $B_i - \{i\}$ is empty or if it contains boundary vertices that are labeled just as leaves, then the partial solution becomes disconnected by adding the new boundary vertex. In such cases we just ignore the entry and move to the next one. Otherwise, we update the value (X',x') of the entry $\mathcal{T}((A',B'),g_p)$, where $B' = (B-\{B_i\})\cup\{\{B_i-i\},\{i\}\}$ and A' is the same as A except it has I in its i-th coordinate, by the value of $(true, \min\{x,x'\})$.

Edge operator $\boxed{i\ j}$*:* If g_p is an edge operator that connects two boundary vertices i and j, then we need to consider two stages. In the first stage we just update the value of each entry $\mathcal{T}((A,B),g_p)$ by the same value as $\mathcal{T}((A,B),g_{p-1})$. This reflects the cases where the edge (i,j) is not used in a partial solution.

Before explaining the next stage we have to mention that only seven different combinations of the labels, when connected together by an edge, make valid partial solutions. These valid combinations are listed in Table 1.

Table 1. Rules for edge operation

Rule	a_i	a_j	a'_i	a'_j	y
1	H	H	S	S	$\min\{x+b(\{i,j\}),x'\}$
2(a)	H	C	S	H	$\min\{x+b(\{i,j\}),x'\}$
2(b)	H	C	S	S	$\min\{x+b(\{i,j\}),x'\}$
3(a)	H	I	H	L	$\min\{x+l(\{i,j\}),x'\}$
3(b)	H	I	S	H	$\min\{x+b(\{i,j\}),x'\}$
4(a)	C	C	H	H	$\min\{x+b(\{i,j\}),x'\}$
4(b)	C	C	H	S	$\min\{x+b(\{i,j\}),x'\}$
4(c)	C	C	S	S	$\min\{x+b(\{i,j\}),x'\}$
5(a)	C	I	S	H	$\min\{x+b(\{i,j\}),x'\}$
5(b)	C	I	C	L	$\min\{x+l(\{i,j\}),x'\}$
6	S	I	S	L	$\min\{x+l(\{i,j\}),x'\}$
7	I	I	C	H	$\min\{x+b(\{i,j\}),x'\}$

Now let $\mathcal{T}((A,B),g_{p-1}) = (true,x)$ and $A = (a_0,\ldots,a_k)$. We find an entry $\mathcal{T}((A',B'),g_p) = (X',x')$ where A' and B' are as follows. The vector A' has a coordinate $(a'_0,\ldots,a'_k)$ where $a'_t = a_t$ if $t \neq i,j$ with the exceptions that (i) if any of i or j has an H label in A that belongs to a star, then we also change the C label of the center of the star in A to an H label in A', and (ii) if any of i or j has a C label that is changed to an H, then the H label in the corresponding star in A is replaced by an L label in A'. The values of a'_i, a'_j are determined by finding a proper match in a row of Table 1 with respect to the values of a_i and a_j.

Also $B' = (B \setminus \{B_i, B_j\}) \cup \{B_i \cup B_j\}$, where B_i and B_j are the elements of B that contain i and j, respectively. We update the value of $\mathcal{T}((A', B'), x')$ by $(true, y)$, where y is the corresponding value given in Table 1.

Boundary Join Operator $H \oplus H'$: Let g_p be a boundary join operator that unifies the boundary vertices of two k-parses $H = (h_0, \ldots, h_r)$ and $H' = (h'_0, \ldots, h'_s)$, where H and H' are substrings of G. For the sake of simplicity we suppose that they have two different tables, $\mathcal{T}_H$ and $\mathcal{T}_{H'}$. We save the result of the $H \oplus H'$ operation in a new table $\mathcal{T}_\oplus$, whose first-column entries are initialized by $(false, \infty)$.

Let $\mathcal{T}_H((A, B), h_r) = (true, x)$ and $\mathcal{T}_{H'}((A', B'), h'_s) = (true, x')$ be two *true* entries in the last columns of $\mathcal{T}_H$ and $\mathcal{T}_{H'}$. If there are two different boundary values i and j that they belong to the same partition (connected components) in both B and B', then this operator creates a cycle which we ignore. In such cases we move to the next possible pair of *true* entries. Otherwise, we find an entry $\mathcal{T}_\oplus((A'', B''), \oplus) = (X'', x'')$, where for each coordinate i in $A'' = (a''_0, \ldots, a''_k)$ there is a match for a_i, a'_i, and a''_i in a row of Table 2 and

$$B'' = \{B_i \cup B'_i \mid i \in \partial, B_i \in B \text{ and } B'_i \in B'\}.$$

Then we update the value of the entry by $(true, \min\{x + x', x''\})$.

We follow that procedure until we update all entries in the last column. Then among all *true* entries in the last column of $\mathcal{T}$ that has a partition set the same as $\{\partial\}$, anyone with the smallest cost is a minimum spanning caterpillar of G.

Table 2. Rules for boundary join operation

Rule	a_i	a'_i	a''_i
1	S	$\{H, I, C\}$	S
2	H	$\{I, C\}$	H
3	H	H	S
4	C	$\{C, I\}$	C

4 Correctness of the Algorithm

In this section we justify the correctness of our minimum spanning caterpillar algorithm. We first show that if there is a *true* entry in the last column of a state table, then the graph has a spanning caterpillar.

Lemma 1. *Let $G = (g_0, \ldots, g_m)$ be a graph whose $tw(G) = k$ and also let $\mathcal{T}$ be the table produced by the algorithm. If $\mathcal{T}((A, B), g_m)$ has a true entry in the last column of $\mathcal{T}$ such that $B = \partial$, then the graph G has a spanning caterpillar.*

Proof. We show that each true entry in the column p, $p \leq m$, of $\mathcal{T}$ relates to a partial solution that has the following properties:

1. the partial solution is a forest of caterpillars,
2. the partial solution covers all vertices in $(g_0, \ldots, g_p)$.

The conditions are satisfied for the only true entry in the first column of $\mathcal{T}$. Since in the first step we set the element $\mathcal{T}((A, B), g_k)$ to *true*, where $A = (I, \ldots, I)$ and $B = \{\{0\}, \{1\}, \ldots, \{k\}\}$.

Now suppose that the conditions hold for each true entry in a column $p-1$, $k \leq p-1 < m$. We show that they also hold for each true entry that is produced by the next operation, g_p. The lemma is true when $\mathcal{T}((A', B'), g_p)$ is the resulting *true* entry from $\mathcal{T}((A, B), g_{p-1})$ by a vertex operation. When g_p is an edge operation Property 2 trivially holds, since no new vertex is added. Also each rule for an edge operation maintains the first property. The same argument is applicable when g_p is a boundary join operation.

Since Properties 1 and 2 hold for g_m and also since $B = \{\partial\}$, the partial solution corresponds to $\mathcal{T}((A, B), g_m)$ is a spanning caterpillar for G. □

Now we prove that if a graph has a spanning caterpillar, our algorithm can recognize it. We first prove this claim for the bounded pathwidth graphs, Lemma 2, then we give a proof for the bounded treewidth in Lemma 3.

Lemma 2. *Let $G = (g_0, \ldots, g_m)$ be a graph whose $pw(G) = k$. If G that has a spanning caterpillar then the last column of the table $\mathcal{T}$ that results from the algorithm has a* true *entry $\mathcal{T}((A, B), g_m)$ with $B = \partial$.*

Proof. Without loss of generality we suppose that C is a spanning caterpillar of G such that each leaf of it is attached to a spine vertex with the smallest index in the k-parse representation. We prove this stronger claim by showing that such spanning caterpillar appears as a partial solution represented by an entry *true* in the last column.

We prove the statement by an inductive argument on the number of vertices of the spine of C. If C has only one vertex on its spine then it is a star and the vertex that appears on the center of C takes a unique boundary value in the k-parse, otherwise it fails to attach to all vertices of G. So the center of the star always appears on each boundary and we attach it to a vertex u when u appears on a boundary by a vertex operation.

Now assume the lemma is valid for any k-parse that has a spanning caterpillar with less than p vertices on its spine, $p \geq 2$. Let C be a spanning caterpillar of G that has p vertices on its spine. Suppose v is the spine vertex that is created by g_f, the vertex operation that assumes the largest index among all vertex operators corresponding to spine vertices of C. We delete v and all its leaves in C and if v is not a head we connect its two neighbors on the spine by adding an edge. The resulted graph H has a caterpillar D with $p-1$ vertices on its spine. We can consider the k-parse representation of H as $(g_0, \ldots, g_{f-1})$, when v is a head, or $(g_0, \ldots, g_{f-1}).g_e$, where g_e is the edge operation corresponds to attaching the neighbors of v on the spine. Because of our inductive assumption, the last column of the table of the algorithm, when applied to H, has a true entry that its partial solution is D. Note that the table of H is the same as the

table produced by the algorithm when applied to G in the column $f-1$. If v is not a head in C we just discard the edge operation g_e to allow the neighbors of v on the spine to appear as heads in the partial solution. Now as v stays as a boundary vertex during $g_f, \ldots, g_m$, the leaves of C can be attached to v by their appropriate edge operation. □

Lemma 3. *Let $G = H \oplus H'$, where H and H' are graphs with treewidths at most k. If G has a spanning caterpillar then the column of the table $\mathcal{T}$, that results from applying the algorithm to G, has a true entry $\mathcal{T}((A,B), H \oplus H')$ with $B = \partial$.*

Proof. If C is a spanning caterpillar in $G = H \oplus H'$, then $H \cap C$ and $H' \cap C$ are forests of caterpillars that span H and H', respectively. To connect the (spanning) forests of caterpillars in H, we first direct the edges on the spine of C from one head to the other. Then we walk along the path on the spine. Once we leave H (by entering to a non boundary vertex of H') and return to it (by entering to a non-boundary vertex of H), we add an edge between the two consecutive visited boundary vertices. By the same method we connect components of $H' \cap C$. Note that since we join the connected components via their heads, the resulting graphs are spanning caterpillar of H and H'. We consider the new edges as extensions of the k-parses of H and H'.

Now we apply the algorithm to the extended k-parses of H and H'. By Lemma 3 we know that the last column of each table has a true entry. Since the extensions are done by adding edges, there are also *true* entries on the last columns of the tables associated to k-parse representations of H and H'. In particular, there are *true* entries such that their partial solutions are associated to $H \cap C$ and $H' \cap C$, so joining them by an $\oplus$ operator produces a *true* entry that has C as its partial solution. □

Theorem 1. *The algorithm solves the spanning caterpillar problem in time $O(5^{k+1}B_{k+1}n)$ for a graph of bounded pathwidth k and in $O(5^{k+1}B_{k+1}^2 n)$ for a graph of bounded treewidth k; where n is the number of vertices and B_{k+1} is the $(k+1)$th Bell number (the number of partitions of a set with $k+1$ members).*

Proof. Note that each k-parse G has a representation as (a) $G = G' \cdot H$, or (b) $G = G' \oplus G''$, where G' and G'' each has treewidth at most k and H is a sequence of vertex and edge operators. The correctness of the algorithm follows from an inductive argument on the number of operators as in (a) and (b). The validity of the base case is the result of Lemmas 2 and 3. For the induction step one just need to use the same technique as Lemma 3 to reduce a problem to the cases with less number of operations. The optimality of the final solution follows from an inductive argument using the fact that we always choose partial solutions with smaller costs.

To solve the problem for a graph with pathwidth at most k, the algorithm uses a table that has $O(5^{k+1}B_{k+1}n)$ entries. In the case when the graph has bounded treewidth, $tw(G) = k$, the algorithm processes each boundary join operation by comparing all pairs of entries in the last two columns of the joined graphs. So it takes $O(5^{k+1}B_{k+1}^2 n)$ steps. □

5 Related Problems

In this section we briefly explain how one can apply the idea of our algorithm to solve other related problems such as the Minimum Spanning Ring Star Problem and the Dual Cost Minimum Spanning tree.

To solve the MSRSP when the input is a graph and its tree decomposition, we follow almost the same procedure as our algorithm for the MSCP. The main difference is that during an edge operation or a boundary joint operation, we allow a cycle to appear. But afterwards, the other vertices can join the cycle just as *leaves*. So as soon as a cycle appears in a partial solution we check to see if the other connected components are isolated vertices. If so, we keep the cycle and we follow the process. Otherwise, we ignore the cycle and move to the next step. Of course here we need to have appropriate bookkeepings to distinguish between state vectors that represent forests of caterpillars and state vectors that represent cycles.

To solve the Dual Cost Minimum Spanning Tree, we ignore the role of spine vertices and we consider forests of trees as partial solutions rather than forests of caterpillars.

6 Conclusion and Further Work

In this paper we presented an algorithm that efficiently finds a minimum spanning caterpillar in some classes of graphs that have small treewidth, like outerplanar, series-parallel and Halin graphs. Our algorithm can be easily modified to solve other related network problems like the Minimum Ring Star Problem. Here one just need to allow cycles to appear in the process of the algorithm and keep the required information to see if it eventually spans the graph. We also can use the same idea to solve the Dual Cost Minimum Spanning Tree.

Also if the input consists of a graph and its treewidth (instead of a tree decomposition) then by using any algorithm that computes a tree decomposition, our algorithm gives another proof that MSCP is fixed-parameter tractable [6], rather than using an EMSOL expression.

It would be interesting if one can improve the constant factor in the running time of our algorithm. It is also of interest if one can improve the time complexity by reducing the size of a state table by a tradeoff with accuracy.

Acknowledgement. The authors thank Sonny Datt for implementing the algorithm and spotting a couple of errors in the former presentation of Table 1.

References

1. Baldacci, R., Dell'Amico, M., Salazar González, J.: The capacitated-ring-star problem. Operations Research 55(6), 1147–1162 (2007)
2. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)

3. Bodlaender, H.L.: Treewidth: Structure and algorithms. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 11–25. Springer, Heidelberg (2007)
4. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 193–242. Elsevier, Amsterdam (1990)
5. Dinneen, M.J.: Practical enumeration methods for graphs of bounded pathwidth and treewidth. Technical Report CDMTCS–055, Center for Discrete Mathematics and Theoretical Computer Science, Auckland (1997)
6. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
7. Hlinený, P., Oum, S.i., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. Comput. J. 51(3), 326–362 (2008)
8. Labbé, M., Laporte, G., Martín, I.R., González, J.J.S.: The ring star problem: Polyhedral analysis and exact algorithm. Networks 43(3), 177–189 (2004)
9. Robertson, N., Seymour, P.D.: Graph minors. II. algorithmic aspects of tree-width. J. Algorithms 7(3), 309–322 (1986)
10. Simonetti, L., Frota, Y., de Souza, C.C.: An exact method for the minimum caterpillar spanning problem. In: Cafieri, S., Mucherino, A., Nannicini, G., Tarissan, F., Liberti, L. (eds.) CTW, pp. 48–51 (2009)
11. Tan, J., Zhang, L.: The consecutive ones submatrix problem for sparse matrices. Algorithmica 48(3), 287–299 (2007)

Fast Algorithms for MIN INDEPENDENT DOMINATING SET*

Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos

LAMSADE, CNRS and Université Paris-Dauphine, F-75775 Paris, France
{bourgeois,escoffier,paschos}@lamsade.dauphine.fr

Abstract. We first devise a branching algorithm that computes a minimum independent dominating set with running time $O^*(2^{0.424n})$ and polynomial space. This improves the $O^*(2^{0.441n})$ result by (S. Gaspers and M. Liedloff, *A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs*, Proc. WG'06). We then study approximation of the problem by moderately exponential algorithms and show that it can be approximated within ratio $1+\epsilon$, for any $\epsilon > 0$, in a time smaller than the one of exact computation and exponentially decreasing with ϵ. We also propose approximation algorithms with better running times for ratios greater than 3.

1 Introduction

Given a graph $G(V, E)$, an independent set of G is a subset $S \subseteq V$ such that, for any $(v_i, v_j) \in S \times S$, $(v_i, v_j) \notin E$. An independent dominating set is an independent set that is maximal for inclusion. MIN INDEPENDENT DOMINATING SET is known to be **NP**-hard [7]. Furthermore, it is also very hard from an approximation point of view, since no polynomial algorithm can approximately solve it within ratio $|V|^{1-\epsilon}$, for any $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$ [9].

For MIN INDEPENDENT DOMINATING SET, the trivial $O^*(2^{|V|})$ bound has been initially broken by [10] down to $O^*(3^{|V|/3}) = O^*(2^{0.529|V|})$ (notation $O^*(\cdot)$ is used to measure complexity of an algorithm ignoring polynomial factors) using a result by [11], namely that the number of maximal (for inclusion) independent sets in a graph is at most $3^{|V|/3}$. This result has been dominated by [8] where using a branch & reduce technique an algorithm optimally solving MIN INDEPENDENT DOMINATING SET with running time $O^*(2^{0.441|V|})$ is proposed.

In this paper, we first devise a branching algorithm that computes a minimum independent dominating set in general graphs with running time $O^*(2^{0.424|V|})$ and polynomial space. We then tackle approximation of MIN INDEPENDENT DOMINATING SET by moderately exponential algorithms and show that there exist $(1+\epsilon)$-approximations obtained in time $O^*(2^{0.424(1-\epsilon/168)n})$ for every $\epsilon \leqslant 6$; we also propose algorithms with better running times for ratios greater than 3.

* Research supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

B. Patt-Shamir and T. Ekim (Eds.): SIROCCO 2010, LNCS 6058, pp. 247–261, 2010.

In what follows, given a graph $G(V, E)$ and a vertex $v \in V$, we denote by $N(v)$ the neighborhood of v, by $N[v] = N(v) \cup \{v\}$ the closed neighborhood of v, by $d(v) = |N(v)|$ the degree of v. For a subset $H \subset V$, we denote by $G[H]$ the subgraph of G induced by H. For $v \in H$, for some $H \subset V$, we denote by $d'_H(v)$ the degree of v in $G[H]$; when no confusion arises, we simplify notations using $d'(v)$ instead. For convenience, we set $N[H] = \{N[v] : v \in H\}$. We use δ and Δ to denote the minimum and maximum degree of G, respectively. For simplicity, we set $n = |V|$ and $m = |E|$; $T(n)$ stands for the maximum running time an algorithm requires to solve MIN INDEPENDENT DOMINATING SET in a graph containing at most n vertices.

Branch & reduce-based algorithms have been used for decades, and a classical analysis of their running times leading to worst case complexity upper bounds is now well-known. If one knows that computing a solution on an instance of size n amounts to computation of a solution on a sequence of p instances of respective sizes $n - k_1, \ldots, n - k_p$, one can write

$$T(n) \leqslant \sum_{i \leqslant p} T(n - k_i) + q(n) \tag{1}$$

for some polynomial q. The running time $T(n)$ is bounded by $O^*(c^n)$, where c is the greatest root of $1 = \sum_{i \leqslant p} x^{-k_i}$. This root is often called the contribution of the branching to overall complexity factor, or the branching factor. In the sequel, we will omit for simplicity to precise the additive polynomial term $q(n)$ in recurrence relations. Of course, it is possible that there does not exist only one single recurrence as in (1), but several ones, depending on the instance. In this case, the running time is never greater than what is needed to solve an instance where at every step we make a branching that has the highest possible branching factor, i.e., the largest solution of (1). This is actually not true anymore when doing *multiple branchings* such as "either we take a or not, and if we add a to the solution then we know that b has degree 3 in the remaining graph and one can make a very good branching on it ...". Indeed, it might be the case that the global worst case of the multiple branching does not correspond to the worst cases of the single branchings it involves. In the sequel, we keep this remark in mind in order to compute worst case running times involving multiple branchings.

2 General Recurrence

Following an idea by [8], we partition the graph into "marked" and "free" vertices. Marked vertices are those that have already been disqualified from belonging to optimum, but still remain non-dominated. Indeed, we generalize the problem at hand in the following way: "given a subset $W \subseteq V$ (W is the set of free vertices), find the minimum independent set in W that dominates V". Notice that, without further hypothesis on W, this problem may have no solution (for instance, when $V \setminus W$ contains a vertex and its whole neighborhood); in this case,

we set $\text{opt}(G, W) = \infty$, where $\text{opt}(G, W)$ denotes the value of the optimum for the problem in inverted commas just mentioned.

In what follows, two vertices v and u are said to be equivalent if $N[u] = N[v]$. In this case, we can remove one of them from the graph (a marked one if any).

Let us first consider a very simple branching, on a vertex v of minimum degree $d(v) = \delta$. We can always suppose that $\delta \geqslant 1$, since (not marked) isolated vertices must be added to the solution. Our solution has to dominate v; hence, at least one vertex of $N[v]$ must belong to the solution, and this vertex must be a free one. Furthermore, if some vertex u belongs to the optimum, its neighbors do not so and they can be removed from the graph. Hence, we have the following recurrence:

$$\text{opt}(G) = 1 + \min_{u \in W \cap N[v]} \{\text{opt}(G[V \setminus N[u]])\} \tag{2}$$

By hypothesis, $d(u) \geqslant \delta$, so we get the following inequality:

$$T(n) \leqslant (\delta + 1 - r)T(n - \delta - 1)$$

where r is the number of marked vertices in $N[v]$. Remark that the order we use to examine neighbors u_i, $i = 1, \ldots, \delta$ of v is important, since the branches are not "$v; u_1; u_2; \ldots; u_\delta$" but "$v; \bar{v}u_1; \bar{v}\bar{u_1}u_2; \ldots; \bar{v}\bar{u_1} \ldots u_{\delta-1}u_\delta$" where for a vertex u, $\bar{u}$ means "not u", i.e. u is marked in the corresponding branch (see Figure 1).

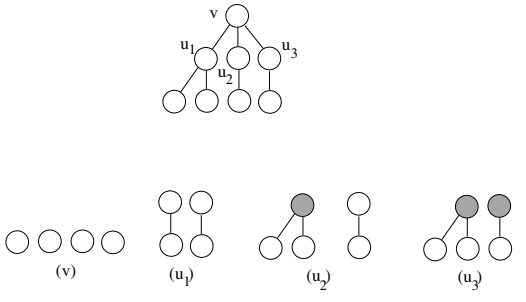


Fig. 1. The four branches $v; \bar{v}u_1; \bar{v}\bar{u_1}u_2; \bar{v}\bar{u_1}\bar{u_2}u_3$. Shaded nodes are marked.

Remark that the complexity of the branching is decreasing with δ, for $\delta \geqslant 2$. So, a straightforward idea is to perform a fine analysis on graphs of low minimum degree. Formally, our algorithm works as follows: (1) if there exists a marked vertex of degree 3 or less, or a vertex which is adjacent to only one free vertex, make a branching according to what is described in Section 3; (2) otherwise, pick a vertex of minimum degree, and branch as described in Section 4.

In the analysis of the running time of the algorithm above, we adopt a measure and conquer approach (see for instance [6]). More precisely, we do not count in the measure the marked vertices of degree at most 2 (they receive weight 0), and we count with a weight $w = 0.2202$ the marked vertices of degree 3. The other vertices count 1 (i.e., they are counted as they are). We so get recurrences on

the time $T(p)$ required to solve instances of size p, where the size of an instance is the sum of the weights of its vertices. Since initially $p = n$, the overall running time is expressed as function of n. This is valid since when $p = 0$, there are only marked vertices in the graph and, in this case, the problem is immediately solved (indeed, there is no solution). Note that this way of measuring progress is introduced in order to simplify the branching analysis (actually, we could obtain a similar result without measure and conquer but with a more technical analysis).

3 Branching on Marked Vertices or Vertices Which Are Adjacent to Only One Free Vertex

Remark first that, if there exists an edge between two adjacent marked vertices, we can remove this edge. This does not increase the complexity measure p.

We branch as follows: if there exist a marked vertex of degree at most 2 or a (free) vertex which is adjacent to only one free vertex (Lemma 1) or, finally, a marked vertex of degree 3 (Lemma 2) then stay in Section 3, otherwise (there are no such vertices), go to Section 4.

Lemma 1. *Assume that some vertex of degree at most* 2 *is marked, or that some free vertex is adjacent to only one free vertex. Then, either the algorithm stops (there is no solution), or we can remove at least one vertex of weight 1 without branching, or, finally,* $T(p) \leqslant T(p-2)+T(p-4)$ *(or a better branching occurs); this branching contributes to the overall complexity with a factor* $\lambda \leqslant 1.2721 = 2^{0.348}$.

Proof. The proof is based upon the study of two cases. In what follows, we call "heavy" a vertex of weight 1 and "light" a vertex of weight w, that means a marked vertex of degree 3.

Case 1. There is some marked vertex v of degree at most 2.
If v has no (free) neighbor, then $\text{opt}(G) = \infty$ (where $\text{opt}(G)$ denotes the value of the optimum in G). If v has exactly one free neighbor u we add u to the solution and we reduce the current instance's size by 1 without branching.

Suppose now that v is marked and that is adjacent to u_1, u_2 (which are free). Then:

$$\text{opt}(G) = 1 + \min\left\{\text{opt}\left(G \setminus N\left[u_1\right]\right), \text{opt}\left(G \setminus N\left[u_2\right]\right)\right\}$$

If both u_1 and u_2 are adjacent to at least 2 heavy vertices, then $T(p) \leqslant 2T(p-3)$, that is better than the result claimed. If some u_i is adjacent only to marked vertices, we must add it to the optimum, decreasing p by 1 without branching. Otherwise, u_1 is adjacent to only one free vertex t_1 and u_2 is adjacent to at least one free vertex t_2. We first suppose that u_1 and u_2 are not adjacent. Then, one of the following situations occurs:

- If $t_1 = t_2$, the only possibility to have both u_1 and v dominated is to add u_1 to the solution; thus we reduce the current instance without branching.
- Otherwise we branch on v. When we add u_2 to the solution, we must add t_1 too, in order to dominate u_1; this leads to $T(p) \leqslant T(p-2) + T(p-4)$.

If none of the former cases happens, then we know that each marked vertex v of degree 2 has two neighbors u_1 and u_2 that are adjacent to each other with at least one of them, say u_1, which is not adjacent to any other free vertex. Then we can remove v without branching, because we already need to take either u_1 or u_2 to dominate u_1. Note that u_1 is adjacent to only one free vertex, so we are in the second case of the lemma.

Case 2. There is a vertex u_1 adjacent to only one free vertex u_2, and there is no marked vertex of degree at most 2.
Note that we can remove any marked vertex which is adjacent to both u_1 and u_2 (such a vertex will be dominated anyway).

We now distinguish the following cases:

(a) u_2 is adjacent to at least two other heavy vertices. We immediately get $T(n) \leqslant T(n-2) + T(n-4)$.
(b) u_2 is adjacent to (exactly) one other heavy vertex t_2.
- If $d(u_1) = 1$, then we branch on t_2 and get at least $T(n) \leqslant 2T(n-3)$ (or even better if t_2 is marked);
- If the sum of the number of light neighbors of u_1 and u_2 is at least 2, we branch on u_2 and get $T(n) \leqslant T(n-2-2w) + T(n-3-2w)$, that leads to $\lambda \leqslant 1.269$ (remember that a light vertex that looses a neighbor has weight 0).
- The only remaining case is when u_1 has exactly one light neighbor v and u_2 has no light neighbor. Then $d(t_2) \geqslant 2$ (or we are in one of the previous cases). If $N(t_2) = \{u_2, v\}$, then we do not need to branch, it is never interesting to take u_1 and t_2 (take u_2 and the third neighbor of v instead). Otherwise, we branch on t_2 and get as previously $T(n) \leqslant T(n-2-2w) + T(n-3-2w)$.

(c) u_2 is adjacent only to light neighbors (and by symmetry u_1 also, or we are in one of the previous cases).
- If u_1 is not adjacent to any other vertex, we can immediately remove it and add u_2 to the solution.
- If u_1 and u_2 are both adjacent to two light vertices, we get $T(n) \leqslant 2T(n-2-4w)$, that leads to branching factor 1.2721 also.
- By symmetry, the only remaining case occurs when u_1 is adjacent to exactly one light vertex v, and u_2 to at least one (other) light vertex. Let x and y the two other neighbors of v. Note that if we take x or y, we can remove u_1 and add u_2. Then, if x and y are both adjacent to at least two free neighbors (see Figure 2), we branch on v, taking either x, y or u_1, and get $T(n) \leqslant 2T(n-5-2w) + T(n-4-2w)$, that leads to a branching factor 1.2412. Otherwise, x has only one free neighbor x' and we branch on x, getting $T(n) \leqslant T(n-4-2w) + T(n-2-w) \leqslant T(n-4) + T(n-2)$.

□

Lemma 2. *Assume some vertex v of degree 3 is marked. Then, at worst, $T(p) \leqslant 2T(p-(3+w)) + T(p-(5+w))$, and the branching factor induced is $\lambda \leqslant 1.339 = 2^{0.421}$.*

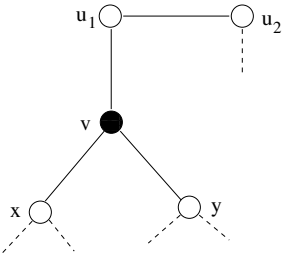


Fig. 2. v is marked. x and y may or may not be adjacent.

Proof. Let $\{u_1, u_2, u_3\}$ be the three neighbors of v. One of the following situations occurs:

1. If each u_i is adjacent to at least 3 free vertices, then, by taking either u_1, u_2 or u_3, we get three branches of size at most $p-(4+w)$.
2. If, say, u_1 is adjacent to two free vertices, then we branch on u_1. In this case, if we take u_1, we reduce p by at least $3+w$. Otherwise, we do not take u_1. In this case u_1 is marked and we can remove the edges between u_1 and the marked vertices. Hence, u_1 and v are marked and have degree at most 2. Then, p reduces by $1+w$. But a further branching on a marked vertex of degree at most 2 (see Lemma 1) allows us either to reduce p by 1, or it creates two branches of size at most $1+w+2=3+w$ or $1+w+4=5+w$ (in the worst case). At worst, $T(p) \leqslant 2T(p-(3+w))+T(p-(5+w))$. □

4 Branching on Vertices of Minimum Degree

We now suppose that the graph does not contain any marked vertex of degree at most 3, and that every vertex is adjacent to at least two free vertices. Then, we branch on the vertex of minimum degree. If this minimum degree is at least 6, then the branching given in Section 2 gives a sufficiently low running time. We distinguish in the following lemmata the different possible values of the minimum degree. Let us start with two preliminary remarks.

Remark 1. When branching on a vertex of minimum degree δ, we can always assume that it is adjacent to at least one vertex of degree at least $\delta+1$. Notice that the branchings performing by the algorithm never increase the degree of a vertex. Then, the situation where the graph is δ-regular occurs at most once (even in case of disconnection). Thus, we make only a finite number of "bad" branchings (where every vertex of minimum degree δ is adjacent only to vertices of degree δ). Such branchings may increase the global running time only by a constant factor. In particular, if $\delta=5$, the branching given in Section 2 gives $T(p) \leqslant 5T(p-6)+T(p-7)$ leading to a branching factor $1.3384 = 2^{0.421}$.

Remark 2. Suppose that we branch on a vertex v which has a neighbor u_1 adjacent to at most three free neighbors. Let us consider the branch where we take u_k not adjacent to u_1 (for $2 \leqslant k \leqslant d(v)$). In this branch, u_1 is marked.

- If $N_W(u_1) \subseteq N_W(u_k)$, then we cannot take u_k and this branch is useless.
- If there is only one vertex t in $N_W(u_1) \setminus N_W(u_k)$, then in this branch we have to take t and we remove at least $d(u_k) + 3$ vertices ($d(u_k) + 1$ by taking u_k, and t and u_1 by taking t).
- Otherwise, u_1 has two other free neighbors t_1, t_2 which are not in $N(u_k)$. In this branch we create a marked vertex (u_1) of degree 2 and hence we reduce p by $d(u_k)+2$. Thanks to Lemma 1, a further branching on the created marked vertex of degree at most 2, gives two branches where p reduces by at least $d(u_k) + 2 + 2$ and $d(u_k) + 2 + 4$, the other possible branchings of Lemma 1 always leading to better recurrences.

In all, either we have one branch with a reduction of $d(u_k) + 3$, or two branches with $d(u_k) + 4$ and $d(u_k) + 6$ (the latter will always be the worst case).

We are ready now to state the main result of the paper, expressed by the following theorem.

Theorem 1. MIN INDEPENDENT DOMINATING SET *can be solved using polynomial space in time* $O^*(2^{0.424n}) = O^*(1.3413^n)$.

The proof of Theorem 1 is immediate consequence of putting together Lemmata 3, 4 and 5 below settling the cases of minimum degree 2, 3 and 4, respectively.

Lemma 3. *If there exists* $v \in V$ *such that* $d(v) = 2$*, then in the worst case we get* $T(p) \leqslant T(p-3) + T(p-4) + T(p-6) + T(p-8)$ *and the branching factor induced is* $\lambda \leqslant 1.3384 = 2^{0.421}$.

Proof. Set $N(v) = \{u_1, u_2\}$. One of the following situations occurs (note that, according to Remark 1, we can assume that either u_1 or u_2 has degree at least 3):

1. If $d(u_2) \geqslant 4$ and $d(u_1) \geqslant 3$ then by taking either v, or u_1, or u_2, we get three branches of size at most $p-3$, $p-4$ and $p-5$.
2. If $d(u_2) \geqslant 4$ and $d(u_2) = 2$ then, when we take u_2 (u_1 having already been discarded), we must also add the only remaining neighbor w of u_1 to the solution. Thus, thanks to Remark 2 (first and second item), we get $T(p) \leqslant 2T(p-3) + T(p-7)$.
3. Suppose that u_1 and u_2 have degree 3 and they are adjacent. Let t be the third neighbor of u_1 (t is not adjacent to u_2, otherwise u_1 and u_2 are equivalent and we can remove one of them). When taking v, we can also take t since, otherwise, it is useless to take v. Hence, we get three branches, each of size at most $p-4$ (even better on the first branch).
4. If $d(u_1) = 3$, then u_1 and u_2 are not adjacent (either because the case has been dealt before, or because $d(u_2) = 2$ and u_2 would be equivalent to v). Then by branching on v, either we take v (size at most $p-3$), or we take u_1 (size at most $p-4$), or we take u_2 and we do not take v and u_1. According to Remark 2, this last choice reduces p either by $d(u_2) + 3 = 5$, or gives rise to two branches of size at most $p-6$ and $p-8$. □

Lemma 4. *If there exists $v \in V$ such that $d(v) = 3$, then we get at worst $T(p) \leqslant T(p-4) + 3T(p-5)$ and the branching factor induced is $\lambda \leqslant 1.3413 = 2^{0.424}$.*

Proof. Set $N(v) = \{u_1, u_2, u_3\}$. If there exists such a vertex v which is marked, then we only have to consider three branches where p reduces by at least 4; hence, $T(p) \leqslant 3T(p-4)$. The same holds if one of the neighbors of v is marked.

We now consider that neither v nor u_i's, $i = 1, 2, 3$, are marked. If $4 \leqslant d(u_i)$ (for $i = 1, 2, 3$) by branching on v we get one branch of size $p-4$ and three branches of size at most $p-5$. Assume that u_1 has degree 3 and that u_3 has degree at least 4. Note that u_1 cannot be adjacent to both u_2 and u_3, otherwise it is equivalent to v. We consider the three following cases with respect to $N(v)$: either it contains two edges, or just one, or zero edges.

1. $N(v)$ contains two edges. Then wlog., u_3 is adjacent to both u_1 and u_2 (and u_1 is not adjacent to u_2). We get four branches of size at most $p-4$, $p-4$, $p-(d(u_2)+2)$ (since u_1 becomes marked and of degree at most 2) and $p-(d(u_3)+1)$. In the third branch thanks to Remark 2, either we remove one more vertex, or we get two branches of size at most $p-(d(u_2)+4) \leqslant p-7$ and $p-(d(u_2)+6) \leqslant p-9$. We now distinguish two cases with respect to $d(u_3)$.
   (a) $d(u_3) \geqslant 5$. Then, at worst, $T(p) \leqslant 2T(p-4)+T(p-7)+T(p-9)+T(p-6)$.
   (b) $d(u_3) = 4$. Let t be the fourth neighbor of u_3. In the branch we take v we can take also t (indeed, if we do not take it, it is useless to take v since taking u_3 is always better). Hence, we ensure at least one more deleted vertex when taking v getting so $T(p) \leqslant T(p-5) + T(p-4) + 2T(p-5)$.
2. $N(v)$ contains at most one edge.
   (a) Assume first that there is a triangle of vertices of degree 3, say, v, u_1, u_2. Let t_1 and t_2 be the third neighbors of u_1 and u_2, respectively. If two vertices among u_3, t_1 and t_2 are equal (i.e., the same vertex) or adjacent, then either two vertices in the triangle are equivalent (case of equality), or one vertex in the triangle v, u_1, u_2 must belong to the solution because, otherwise, one of them would not be dominated (case of adjacency). Hence, $T(p) \leqslant 3T(p-4)$. Finally, if t_1, t_2 and u_3 are distinct and non adjacent, let us set $\Gamma' = N(t_1) \cup N(t_2) \cup N(t_3) \setminus \{v, t_1, t_2, u_1, u_2, u_3\}$. Either we take one vertex of the triangle, or we have to take t_1, t_2 and u_3, that is, $T(p) \leqslant 3T(p-4) + T(p-6-|\Gamma'|)$ (see Figure 3).
      – If $|\Gamma'| \geqslant 4$, then $T(p) \leqslant 3T(p-4) + T(p-10)$ leading to $\lambda \leqslant 2^{0.417}$.
      – If $|\Gamma'| = 3$, then u_3 has degree 4 and $\Gamma' \subset N(u_3)$. Then, we branch as follows: either we take u_3 and remove 9 vertices, or we take v and remove 4 vertices, or we mark u_3 and v. By removing the edge between them, they become, respectively, of degree 3 and 2. Hence, $T(p) \leqslant T(p-9) + T(p-4) + T(p-(2-w))$ and $\lambda \leqslant 2^{0.411}$.
   (b) Otherwise, if the only edge in $N(v)$ is (u_2, u_3), then we get $T(p) \leqslant T(p-4) + T(p-4) + T(p-5) + T(p-6)$; but in the two last branches, thanks to Remark 2, either we remove one more vertex or we get two branches of size reduced by 2 and 4. So, we get at worst $T(p) \leqslant 2T(p-4) + T(p-7) + T(p-9) + T(p-8) + T(p-10)$.

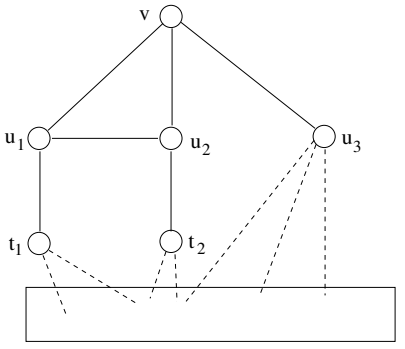


Fig. 3. I' is the rectangular box

(c) If there exists an edge (u_1, u_2) with $d(u_2) \geqslant 4$ (otherwise, we are in case 2a above) we get $T(p) \leqslant T(p-4)+T(p-4)+T(p-5)+T(p-6)$; but, in the last branch, thanks again to Remark 2, we get at worst $T(p) \leqslant 2T(p-4)+T(p-5)+T(p-8)+T(p-10)$.

3. Assume finally that there is no edge in $N(v)$. If u_2 and u_3 have degree at least 4, then we get $T(p) \leqslant T(p-4)+T(p-4)+T(p-6)+T(p-6)$ (indeed, in the last two branches u_1 is marked and has degree at most 2). If, say, u_2 has degree 3, then we get $T(p) \leqslant T(p-4)+T(p-4)+T(p-5)+T(p-6)$; but, in the two last branches, thanks to Remark 2, either we remove one more vertex, or we get two branches of size reduced by 2 and 4. So, at worst, $T(p) \leqslant 2T(p-4)+T(p-7)+T(p-9)+T(p-8)+T(p-10)$. □

Lemma 5. *If there exists $v \in V$ such that $d(v) = 4$ then, at worst $T(p) \leqslant 4T(p-5)+T(p-9)$ with a contribution to the overall branching factor bounded above by $1.3394 = 2^{0.422}$.*

Proof. Set $N(v) = \{u_1, u_2, u_3, u_4\}$ and assume that u_4 has degree at least 5. If at least three out of four u_i's have degree at least 5, then $T(p) \leqslant 2T(p-5) + 3T(p-6)$. Consider now that u_1 and u_2 have degree 4.

Suppose first that the u_i's of degree 4 are not adjacent. In the branch we take u_2, p reduces by $6-w$ (since u_1 is marked and has degree at most 3). Then, either u_3 has degree at least 5 and then $T(p) \leqslant 2T(p-5)+T(p-(6-w))+2T(p-6)$, or u_3 has degree 4 and in this case, in the branch we take u_3, p reduces by $5+2(1-w)$. In all, $T(p) \leqslant 2T(p-5)+T(p-(6-w))+T(p-(7-2w))+T(p-6)$ and $\lambda \leqslant 2^{0.414}$.

Assume now that u_1 and u_2 are adjacent. When v, u_1 and u_2 are marked, they become of degree 2. Then:

- If u_1 and u_2 are not adjacent to u_3, $T(p) \leqslant 3T(p-5)+T(p-7)+T(p-6)$.
- If u_1 is not adjacent to u_3 and u_2 is not adjacent to u_4, we get $T(p) \leqslant 3T(p-5)+T(p-6)+T(p-7)$.
- Otherwise, both u_1 and u_2 are adjacent to u_3 but not to u_4. If u_3 has degree at least 5, $T(p) \leqslant 3T(p-5)+T(p-6)+T(p-7)$. Otherwise, u_4 is not

adjacent to any of the u_i's (or one of them would be equivalent to v). Then, we get 4 branches of size $p-5$ and in the last branch the u_i's are marked and have degree at most 2 hence p reduces by 9: $T(p) \leqslant 4T(p-5)+T(p-9)$. □

5 Approximation of MIN INDEPENDENT DOMINATING SET by Moderately Exponential Algorithms

As we have mentioned in Section 1, for any $\varepsilon > 0$, MIN INDEPENDENT DOMINATING SET is inapproximable within ratio $n^{1-\varepsilon}$ unless $\mathbf{P} = \mathbf{NP}$. On the other hand, it is easy to see that any maximal independent set guarantees a ratio at most $\Delta + 1$. In this section, we devise algorithms achieving ratios "forbidden" in polynomial time (such as constant ratios), with running times that, although exponential, are better than the running time of exact computation for MIN INDEPENDENT DOMINATING SET. Such a problematic has already been tackled for other optimization problems such as Maximum Independent Set, Minimum Set Cover, Min Coloring (see for instance [2,3,5]). A first question is whether we can provide or not a family of algorithms with approximation ratio $1+\epsilon$ (for *any* $\epsilon > 0$) in time $O^*(\gamma_\epsilon^n)$ where $\gamma_\epsilon < 2^{0.424}$, a kind of exponential approximation scheme. This is quite easy for some problems such as hereditary problems ([2]) but seems harder for other problems such as MIN INDEPENDENT DOMINATING SET. We answer to this question in Proposition 1. Then, we improve this result for bounded degree graphs (Proposition 2). Finally, we propose another algorithm leading to improved running times for ratios greater than 3 (Proposition 4).

The algorithms presented in this section use the following lemma by [4].

Lemma 6. *([4]) For any $k \geqslant 3$, it is possible to enumerate all independent dominating sets (i.e., maximal independent sets) of size at most n/k with running time $O^*(k^{n/k})$.*

Our first result is given in the following Proposition 1 and claims that there exist $(1+\epsilon)$-approximation algorithm that runs in time $O^*(2^{0.424(1-\epsilon/168)n})$.

Proposition 1. *For any positive $\epsilon \leqslant 6$,* MIN INDEPENDENT DOMINATING SET *is $(1+\epsilon)$-approximable in time $O^*(2^{0.424(1-\epsilon/168)n})$.*

Proof. The algorithm claimed, denoted by `IDS`, works as follows. It first computes all the dominating independent sets of size $n/7$ or less, with running time $O^*(7^{n/7}) = O^*(2^{0.402n})$ (Lemma 6). If such a set exists, it returns a smallest among them. Otherwise, $\mathrm{opt}(G) \geqslant n/7$. In this case, it sets $r = 0$ and then, while the graph is not empty and $r \leqslant \epsilon n/168$, it repeats the following steps:

1. pick a vertex v of minimum degree (say δ); if $\delta \geqslant 6$, branch on v as discussed previously;
2. otherwise, if $\sum_{u \in N(v)}(d(u)-1) \geqslant 24$, branch on v;
3. otherwise, add v to the solution and remove its neighbors and increase r by 1.

Finally, when r reaches $\epsilon n/168$ it runs the exact algorithm on the remaining graph denoted by $G'(V', E')$.

Each time step 1 is applied, algorithm IDS makes a branching whose recurrence is $T(n) \leqslant (\delta+1)T(n-(\delta+1))$, with $\delta \geqslant 6$. On the other hand, each time step 2 is applied, branching's recurrence is $T(n) \leqslant 4T(n-5)+T(n-16)$ or better. Indeed, if $d(v) = 5$ then $\sum_{u\in N(v)} d(u) \geqslant 29$, and the worst case occurs when four neighbors of v have degree 5 and one has degree 9 leading to a recurrence $T(n) \leqslant 5T(n-9)+T(n-16)$. Similarly, if $d(v) = 4$, $d(v) = 3$, $d(v) = 2$ or $d(v) = 1$ the worst cases give respectively the recurrences $T(n) \leqslant 4T(n-5)+T(n-17)$, $T(n) \leqslant 3T(n-4)+T(n-22)$, $T(n) \leqslant 2T(n-3)+T(n-25)$ and $T(n) \leqslant T(n-2)+T(n-26)$. The worst case occurs when v has degree 4, leading to a branching factor $2^{0.403}$. If the algorithm IDS stops because G is empty, then the global running time is $O^*(2^{0.403n})$, that is better than the time claimed in the proposition's statement (since for $\epsilon \leqslant 6$, $0.424(1-\epsilon/168) > 0.403$).

Assume now that $r = \epsilon n/168$, and let q be the number of times steps 1 or 2 have been run. Clearly, the larger the q, the faster the algorithm. Thus, the global running time is

$$T(n) \leqslant 2^{0.424(n-p)} \leqslant 2^{0.424n(1-\epsilon/168)}$$

Denote by S the solution computed by IDS. We prove that $|S|$ is bounded above by $(1+\epsilon)\,\mathrm{opt}(G)$. Let S^* be an optimal solution. The algorithm builds a branching tree corresponding to branching steps 1 and 2. Note that, unfortunately, due to step 3, when branching on a vertex v, it might be the case that no vertex in $N[v]$ belongs to S^*. Indeed, v might be dominated by a vertex deleted from the graph (a neighbor of a vertex added in step 3). However, in such a tree, consider the following path (starting from the root). At a node of the search tree dealing with vertex v:

- if S^* has (at least) one vertex in $N[v]$, follow this branch (one of these branches);
- otherwise, follow the branch where v is added in the solution.

Following this path, one reaches a particular leaf where one applies the exact algorithm.

Now let Ω be the set of vertices we arbitrarily added to the solution during executions of step 3 and $K = N[\Omega]$. Denote by B the set of vertices added during steps 1 and 2 (following the path previously described) and set $B_1 = B \cap S^*$ and $B_2 = B \setminus B_1$. Set $S_1^* = S^* \cap K$, $S_2^* = S^* \cap N[B]$ and $S_3^* = S^* \cap V'$. Set, finally, $q_1 = |B_1|$ and $q_2 = |B_2|$ and note that $(K, N[B], V')$ is a partition of V.

Fact 1. $S^* \cap (N[B] \setminus B) = \emptyset$.

Indeed, this comes from the path followed in the tree. In other words, $S_2^* = B_1$.

Fact 2. $B_2 \subseteq N(S_1^*)$.

For Fact 2, consider the search tree's node where v is added in B. Since $v \notin B_1$, no neighbor of v (in the current graph) is in S^*. So, there exists a neighbor of v in S^* deleted previously from the graph because of step 3. This vertex is in S_1^*.

Thanks to Fact 1, $V' \cap N[S_2^*] \subseteq V' \cap N[B] = \emptyset$. This means that each vertex in V' is either in $N[S_3^*]$ or in $N[S_1^*]$. Then, there exists an independent dominating set of $G[V']$ included in $S_3^* \cup (N[S_1^*] \cap V')$, meaning that:

$$\mathrm{opt}\,(G\,[V']) \leqslant |S_3^*| + |N\,[S_1^*] \cap V'| \tag{3}$$

On the other hand, thanks to Fact 2, $|N(S_1^*) \setminus K| \geqslant q_2 + |N[S_1^*] \cap V'|$. Moreover:

$$|N\,(S_1^*) \setminus K| \leqslant |N(K) \setminus K| \leqslant \sum_{v \in \Omega} \sum_{u \in N(v)} (d(u) - 1) \leqslant 23r \tag{4}$$

Combining (3) and (4), we get $\mathrm{opt}(G[V']) + q_2 \leqslant |S_3^*| + 23r$. The solution S computed by IDS satisfies $|S| = r + q_1 + q_2 + \mathrm{opt}(G[V']) = r + |S_2^*| + |S_3^*| + 23r \leqslant |S^*| + 24r$. Putting all this together we obtain

$$|S| \leqslant |S^*| \left(1 + \frac{24\epsilon n}{168\,|S^*|}\right) \leqslant |S^*|\,(1 + \epsilon)$$

as claimed. □

In what concerns the running time, the result of Proposition 1 can be improved towards various directions. First, in the following Proposition 2 we improve running time for graphs of bounded degree.

Proposition 2. *If the maximum degree of G is bounded above by some constant Δ, then for any $\epsilon > 0$, it is possible to compute a $1+\epsilon$-approximation with running time $O^*(\gamma^{(1-f(\epsilon))n})$, where $f(\epsilon)$ is solution of:*

$$\frac{f(\epsilon)\Delta}{\epsilon} \log\left(\frac{\epsilon}{\Delta f(\epsilon)}\right) = (1 - f(\epsilon)) \log \gamma \tag{5}$$

Proof. Fix some $\epsilon > 0$ and consider the following algorithm: (1) search for any independent dominating set of size at most $f(\epsilon)\Delta n/\epsilon$; if some dominating set is computed, return it as solution (otherwise, $|S^*| > f(\epsilon)\Delta n/\epsilon$); (2) fix an arbitrary vertex-subset K of size $f(\epsilon)n$; (3) run an exact algorithm for MIN INDEPENDENT DOMINATING SET in $G[V \setminus K]$ and let S_1 be the solution computed; (4) run some polynomial algorithm for MIN INDEPENDENT DOMINATING SET in $G[V \setminus N[S_1]]$ and let S_2 be the solution computed; (5) output $S_1 \cup S_2$.

According to (5) and Lemma 6, the running time of the algorithm above is at most:

$$O^*\left(\left(\frac{\epsilon}{\Delta f(\epsilon)}\right)^{\frac{f(\epsilon)\Delta n}{\epsilon}} + \gamma^{n-|K|}\right) = O^*\left(\gamma^{n(1-f(\epsilon))}\right)$$

Set $I = \mathrm{opt}(G[V \setminus (K \cup N(K \cap S^*))])$; I is an independent set and there exists a set I', $V \setminus K \supseteq I' \supseteq I$, that is dominating in $G[V \setminus K]$. Since $S^* \setminus K$ is an independent dominating set (possibly not maximal) in $G[V \setminus (K \cup N(K \cap S^*))]$, the size of S_1 is bounded above by:

$$|S_1| \leqslant |I'| \leqslant |I| + |N\,(K \cap S^*) \setminus K| \leqslant |S^* \setminus K| + |N\,(K \cap S^*) \setminus K|$$

from what we get:

$$\begin{aligned} |S_1 \cup S_2| &\leqslant |S_1| + |K| \quad \leqslant \quad |S^*| + |K| - |K \cap S^*| + |N(K \cap S^*) \setminus K| \\ &\leqslant |S^*| + |K| + (\Delta - 1)\,|K \cap S^*| \quad \leqslant \quad |S^*|\,|K|\Delta \end{aligned}$$

Hence, in all:

$$\frac{|S_1 \cup S_2|}{|S^*|} \leqslant 1 + \frac{\Delta f(\epsilon) n}{\Delta f(\epsilon) n/\epsilon} \leqslant 1 + \epsilon$$

as claimed. □

Table 1 shows the tradeoffs claimed by Proposition 2 between ratio and running time for some values of Δ and for $\gamma = 2^{0.424}$. The entries in the leftmost column are values of ϵ; the other entries are the multipliers of n in the exponent of 2. For instance, if $1 + \epsilon = 1.5$ is to be attained in a graph of maximum degree 5, the running time of the algorithm is $O^*(2^{0.397n})$.

Table 1. Some results derived from Proposition 2. The polynomial case is due to [1].

ϵ	$\Delta = 3$	$\Delta = 4$	$\Delta = 5$	$\Delta = 6$	$\Delta = 7$	$\Delta = 8$	$\Delta = 9$	$\Delta = 10$
0.1	0.415	0.418	0.419	0.420	0.421	0.421	0.422	0.422
0.2	0.406	0.411	0.414	0.415	0.417	0.418	0.419	0.419
0.5	0.376	0.389	0.397	0.400	0.405	0.407	0.409	0.411
1	**P**	0.349	0.366	0.376	0.384	0.389	0.393	0.397
1.5		0.304	0.332	0.349	0.361	0.370	0.376	0.381
2		0.254	0.294	0.320	0.337	0.349	0.358	0.366

Finally, improved running times can be obtained if one wishes to attain "worse" approximation ratios. We give a very simple algorithm leading to Proposition 3, that we slightly improve in Proposition 4.

Proposition 3. *For any $r \geqslant 3$, it is possible to compute an r-approximation of* MIN INDEPENDENT DOMINATING SET *with running time $O^*(2^{n \log_2 r/r})$.*

Proof. We run the branching algorithm of the proof of Lemma 6 to compute every independent dominating set of size n/r or less. If it finds some independent dominating set, it returns it; otherwise, $\mathrm{opt}(G) \geqslant n/r$, where $\mathrm{opt}(G)$ denotes the size of a minimum independent dominating set. In the first case, the algorithm needs time $O^*(r^{n/r}) = O^*(2^{n \log_2 r/r})$ and computes an optimal solution; in the second case, any maximal independent set is an r-approximation and such a set is computed in polynomial time. □

The following proposition further improves the result of Proposition 3.

Proposition 4. *For any $r \geqslant 3$, it is possible to compute an approximation of* MIN INDEPENDENT DOMINATING SET *with running time $O^*(2^{n \log_2 r/r})$ and approximation ratio $r - ((r-1)/r) \log_2 r$.*

Proof. As previously, we first compute every independent dominating set of size n/r or less. If such sets exist, we return one of those with minimum size. Otherwise, we partition V into $l = r/\log_2 r$ subsets $V_1, \ldots, V_l$, of size $n \log_2 r/r$, and we initialize S with some independent dominating set. Then, for $j \leqslant l$, we run the following procedure: (1) for any $H \subseteq V_j$, if H is an independent set, compute an independent dominating set S_H in $G[V \setminus N[H]]$; (2) return the smallest set among S and the sets $H \cup S_H$ computed.

Obviously, S is an independent dominating set. The algorithm examines $l \times 2^{n/l}$ subsets, that concludes the running time claimed.

Fix some optimal solution S^*. Since S^* is maximally independent, $\cup_{i \leqslant l} N(V_i \cap S^*) = V \setminus S^*$ and we get (I.S. stands for independent set):

$$\frac{|S|}{\mathrm{opt}(G)} \leqslant \min_{j \leqslant l} \min_{H \text{ I.S. of } V_j} \left\{ \frac{n - |N(H)|}{\mathrm{opt}(G)} \right\} \leqslant \min_{j \leqslant l} \left\{ \frac{n - |N(V_j \cap S^*)|}{\mathrm{opt}(G)} \right\}$$
$$\leqslant \frac{n - \frac{n - \mathrm{opt}(G)}{l}}{\mathrm{opt}(G)} \leqslant \frac{\log_2 r}{r} + r - \log_2 r$$

that completes the proof of the proposition. □

Table 2. Tradeoffs between running times and ratios derived by Propositions 3 and 4

Ratio	2	3	4	5	10	20	50
Proposition 3		1.4423^n	1.4143^n	1.3798^n	1.2590^n	1.1616^n	1.0814^n
Proposition 4	1.4403^n	1.3870^n	1.3419^n	1.3077^n	1.2130^n	1.1398^n	1.0749^n

Tradeoffs between running times and ratios are displayed in Table 2. Recall that the exact algorithm given above runs in time $O^*(1.3416^n)$.

References

1. Alimonti, P., Calamoneri, T.: Improved Approximations of Independent Dominating Set in Bounded Degree Graphs. In: D'Amore, F., Marchetti-Spaccamela, A., Franciosa, P.G. (eds.) WG 1996. LNCS, vol. 1197, pp. 2–16. Springer, Heidelberg (1997)
2. Bourgeois, N., Escoffier, B., Paschos, V.T.: Efficient Approximation of Combinatorial Problems by Moderately Exponential Algorithm. In: Proc. of WADS 2009. LNCS, vol. 5664, pp. 507–518. Springer, Heidelberg (2009)
3. Bourgeois, N., Escoffier, B., Paschos, V.T.: Approximation of min coloring by moderately exponential algorithms. Inf. Process. Lett. 109(16), 950–954 (2009)
4. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. Oper. Res. Lett. 32(6), 547–556 (2004)
5. Cygan, M., Kowalik, L., Wykurz, M.: Exponential-time approximation of weighted set cover. Inf. Process. Lett. 109(16), 957–961 (2009)
6. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure and conquer approach for the analysis of exact algorithms. Journal of the ACM 56(5) (2009)

7. Garey, M.R., Johnson, D.S.: Computers and intractability. A guide to the theory of NP-completeness. W. H. Freeman, San Francisco (1979)
8. Gaspers, S., Liedloff, M.: A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 78–89. Springer, Heidelberg (2006)
9. Halldórsson, M.M.: Approximating the minimum maximal independence number. Inform. Process. Lett. 46, 169–172 (1993)
10. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Inform. Process. Lett. 27, 119–123 (1988)
11. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. of Mathematics 3, 23–28 (1965)

Author Index