SIROCCO 8
THE 8TH INTERNATIONAL COLLOQUIUM ON STRUCTURAL INFORMATION & COMMUNICATION COMPLEXITY

FRANCESC COMELLAS, JOSEP FÀBREGA, AND PIERRE FRAIGNIAUD
EDITORS
Canadian Cataloguing in Publication Data

International Colloquium on Structural Information and
Communication Complexity (8th : 2001 : Vall de Núria, Spain)

Sirocco 8 : proceedings of the 8th International Colloquium on Structural
Information and Communication Complexity, Vall de Núria, Spain

(Proceedings in informatics ; 10)
Includes bibliographical references.
ISBN 1-894145-10-0

1. Computer networks--Congresses. 2. Electronic data processing--Distributed
processing--Congresses.  I. Comellas, Francesc  II. Title.  III. Series.
# Table of Contents

Francesc Comellas, Josep Fàbrega, & Pierre Fraigniaud  
Preface  .........................................................  v

J. Diaz  
Adversarial Queueing Theory  .......................................  1

C. Kaklamanis  
Wavelength Routing in WDM Trees  ..................................  3

L. Barrière & S. Dobrev  
Leader Election in Abelian Cayley Graphs  .........................  5

J-C. Bermond, L. Chacon, D. Coudert, & F. Tillerot  
Cycle Covering ........................................................  21

P. Bose & P. Morin  
Competitive Online Routing in Geometric Graphs  ..................  35

S. D. Bruda & S. G. Akl  
Parallel Real-Time Complexity: A Strong Infinite Hierarchy  ..  45

S. Cantarella, A. K. Datta, F. Petit, & V. Villain  
Group Mutual Exclusion in Token Rings  ............................  61

I. Caragiannis, C. Kaklamanis, & P. Kanellopoulos  
New Bounds on the Size of the Minimum Feedback Vertex Set in  
Meshes and Butterflies .................................................  77

Victor Chepoi & Alexis Rollin  
Interval Routing in Some Planar Quadrangulations  .................  89

Peter Damaschke  
Worst-Case Bounds for Blind Broadcasting in Small-Degree  
Networks ...............................................................  105

M. Elkin & D. Peleg  
The Client-Server 2-Spanner Problem and Applications to  
Network Design ..........................................................  117

G. Fertin & A. Raspaud  
k–Neighborhood Broadcasting ...........................................  133

P. Flocchini, G. Prencipe, N. Santoro, & P. Widmayer  
Pattern Formation by Autonomous Robots without Chirality  ....  147

Yashar Ganjali  
Characterization of Networks Supporting Multi-dimensional  
Linear Interval Routing Scheme ........................................  163
E. Godard, Y. Métivier
A Characterization of Classes of Graphs Recognizable by
Local Computations with Initial Knowledge .......................... 179

Peter Gvozdjak & Joseph G. Peters
Modelling Links in Inclined LEO Satellite Networks .................. 195

Paul R. Hafner
On the graphs of McKay-Miller-Širáň ................................. 209

A. Kesselman & Y. Mansour
QoS-Competitive Video Buffering ...................................... 217

Rastislav Královič & Peter Ružička
On Immunity and Catastrophic Indices of Graphs ...................... 231

Martin Makúch & Peter Ružička
On the Complexity of Path Layouts in Bounded Degree
ATM Networks: A Case Study for Butterfly Networks ................. 243

G. Malewicz, A. Russell, & A. Shvartsman
Optimal Scheduling for Disconnected Cooperation .................... 259

Achour Mostéfaoui, Sergio Rajsbaum, Michel Raynal & Matthieu Roy
Efficient Condition-Based Consensus .................................. 275

T. Nakata & M. Yamashita
Some Upper Bounds on Expected Agreement Time of a
Probabilistic Local Majority Polling Game ............................ 293

A. Osterloh
Oblivious Routing on $d$-dimensional Meshes .......................... 305

P. Salinger & P. Tvrdík
Broadcasting in All-Output-Port Cube-Connected Cycles with
Distance-Insensitive Routing ............................................. 321

John Watkinson, Micah Adler, & Faith E. Fich
New Protocols for Asymmetric Communication Channels ............ 337

Norbert Zeh & Nicola Santoro
On Finding Minimum Deadly Sets for Directed Networks ............ 351
Preface

This volume constitutes the Proceedings of the 8th Colloquium on Structural Information and Communication Complexity (SIROCCO), held at Vall de Núria, Catalonia, Spain, June 27-29, 2001. The SIROCCO series focuses on the relationship between computing and communication, and the study of the factors (e.g., structure, memory, etc.) that are significant for the computability and the communication complexity of problems in distributed systems (from tightly coupled parallel computers to systems of communicating agents).

The twenty-five contributed technical presentations given at the Colloquium were chosen by the Program Committee from thirty-five extended abstracts received by the submission deadline. The Committee based its decisions on the perceived quality and originality of the submissions and their appropriateness to the theme of the Colloquium. The papers collected in this volume are to be considered ongoing research, and authors will publish, in most cases, a complete final version in an archival scientific journal. We would like to thank all authors who submitted papers and the colleagues who helped by providing reviews of these submissions.

Technical presentations were completed by the invited lectures addressed by Josep Diaz (UPC, Barcelona) and Christos Kaklamanis (CTI, Patras), to whom the Committee is specially grateful.

Finally, we wish to thank the several organizations which financially supported SIROCCO.

Francesc Comellas, Josep Fàbrega, and Pierre Fraigniaud
Vall de Núria, June 2001.

SIROCCO 2001 Sponsors

Ministerio de Ciencia y Tecnología. Generalitat de Catalunya (DURSI).
Universitat Politècnica de Catalunya. CNRS and GdR ARP.
SIROCCO 2001

Program Committee

Francesc Comellas, Chair (UPC, Barcelona, Spain)
Stefan Dobrev (Slovak Academy of Sciences, Slovak Rep.)
Danny Dolev (Hebrew U. of Jerusalem, Israel)
Josep Fàbrega, Chair (UPC, Barcelona, Spain)
Paola Flocchini (U. Ottawa, Canada)
Pierre Fraigniaud, Chair (LRI, Orsay, France)
Satoshi Fujita (U. Hiroshima, Japan)
Cyril Gavoille (LaBRI U. Bordeaux, France)
Lefteris Kiousis (CTI, Patras, Greece)
Evangelos Kranakis (U. Carleton, Ottawa, Canada)
Art Liestman (SFU, Burnaby, Canada)
Fabrizio Luccio (U. Pisa, Italy)
Jarooslav Opatrny (Concordia U., Montreal, Canada)
David Peleg (Weizmann Inst., Israel)
Andrzej Proskurowski (U. Oregon, USA)
Sergio Rajsbaum (UNAM, Mexico)
Jop F. Sibeyn (Max-Planck Inst., Germany)
Ramesh K. Sitaraman (U. Massachusetts and Akamai Tech. Inc., USA)
Sunil M. Shende (U. Rutgers, Camden, USA)
Richard Tan (U. Utrecht, Holland)
Walter Unger (U. Aachen, Germany)
Eli Upfal (Brown University, USA)
Ugo Vaccaro (U. Salerno, Italy)
Sebastiano Vigna (U. Milano, Italy)
Imrich Vrto (Slovak Academy of Science, Slovak Rep.)

Conference Chair

David Peleg

Organizing Committee

Francesc Comellas, Chair (UPC, Barcelona, Spain)
Josep Fàbrega (UPC, Barcelona, Spain)
Pierre Fraigniaud (LRI, Orsay, France)
Margarida Mitjana (UPC, Barcelona, Spain)
Xavier Muñoz (UPC, Barcelona, Spain)
Javier Ozón (UPC, Barcelona, Spain)
Adversarial Queueing Theory:
Recent Results
INVITED LECTURE

JOSÉP DIAZ
Universitat Politècnica de Catalunya, Barcelona

Abstract
An important issue in parallel and distributed computing is the efficient dynamic routing of packets in a network. Each injected packet has its own specified path and conflicts are solved according to different types of protocols. Analyzing the performance of routing algorithms, can be translated into questions of stability and delay of packet routing schedules for the various communication networks. A packet routing schedule is define to be stable if the number of packets stays bounded as the system runs over a long period of time. The traditional line of attack on those problems has been to consider stochastic queuing model assumptions. The recent adversarial theory of queuing developed by Borodin et al. avoids making probabilistic assumptions on the arrival of packets, making positive results on stability more robust than in the previous stochastic models. In this talk, we survey recent results about on the stability and non-stability on networks under different greedy queuing policies: Furthest-to-go, nearest-to-go, longest-in-system, shortest-in-system, FIFO, etc.

Keywords
queueing theory, routing

Josép Diaz is with the Departament de Llenguatges i Sistemes Informatics, Universitat Politècnica de Catalunya, Barcelona, Spain. Web: http://www.lsi.upc.es/diaz E-mail: diaz@lsi.upc.es

Wavelength Routing in WDM Trees

INVITED LECTURE

CHRISTOS KAKLAMANIS
University of Patras, Greece

Abstract

We study the problem of allocating optical bandwidth to sets of communication requests in all-optical networks that utilize Wavelength Division Multiplexing (WDM). WDM technology establishes communication between pairs of network nodes by establishing transmitter–receiver paths and assigning wavelengths to each path so that no two paths going through the same fiber link use the same wavelength. Optical bandwidth is the number of distinct wavelengths. Since state-of-the-art technology allows for a limited number of wavelengths, the engineering problem to be solved is to establish communication between pairs of nodes so that the total number of wavelengths used is minimized; this is known as the wavelength routing problem.

We survey recent advances in bandwidth allocation in tree-shaped WDM all-optical networks. We present hardness results and lower bounds for the general problem and the special case of symmetric communication. We give the main ideas of deterministic greedy algorithms and study their limitations. We demonstrate how we can achieve optimal and nearly-optimal bandwidth utilization in networks with wavelength converters using simple algorithms. We also present recent results about the use of randomization for wavelength routing.

Keywords

WDM optical networks, wavelength routing, wavelength conversion

Christos Kaklamanis is with the Computer Technology Institute and Dept. of Computer Engineering and Informatics, University of Patras, 26500 Rio, Greece. E-mail: kkl@cti.gr
Leader Election in Abelian Cayley Graphs*

L. Barrière
Univ. Politècnica de Catalunya, Barcelona

S. Dobrev
Slovak Academy of Sciences, Bratislava

Abstract

In this paper we consider the problem of distributed leader election in unoriented Abelian Cayley graphs of degree 4. This family of graphs, denoted by $C\mathcal{G}_4$, include bidimensional tori of arbitrary size and chordal rings with one arbitrary chord. We present an algorithm which assumes that the only structural knowledge available to the nodes is “I am in an Abelian Cayley Graph of degree 4”. The message complexity of our algorithm, for a graph of $C\mathcal{G}_4$ of order $N$ and diameter $D$, is $O(N + D \log \frac{D}{N})$.

The obliviousness of our algorithm to the underlying topology has been achieved by studying the properties of tessellations of the plane and reducing the problem of leader election in a graph from $C\mathcal{G}_4$ to election in the associated tessellation.

Keywords
distributed computing, distributed algorithms, leader election, Cayley graphs, plane tessellations

1 Introduction

Identification and study of the factors that strongly influence solvability and complexity of problems is one of the main research areas in the field of distributed computation. Over the course of years, the importance of structural information, that is, the a priori knowledge available to the nodes about the structure of the system, has become apparent. In particular, two types of structural information, topological awareness and sense of direction (SD), have been identified in [19] as having strong impact on the complexity of distributed algorithms. Topological awareness means that the nodes are aware of the class of networks they are in, for instance being in a hypercube. Sense of direction has been introduced in [19].

---

*This work has been done while the authors were visiting the Department of Computer Science of Carleton University, Ottawa. Additional support by Spanish Research Council (CICYT) under project TIC2000-1017 and NATO and Slovak VEGA grant no. 2/7007/20
as a globally consistent labelling of communication ports in the nodes. There are several types of SD, ranging from the uniform group-based SD, which is the strongest one, covering for instance dimensional SD in hypercubes and compass SD in tori [20], to SD and Weak SD as defined in [10].

This paper contributes to the study of the impact of structural information on the message complexity of distributed leader election – moving the system from an initial configuration where the nodes are in the same state to a final configuration with exactly one node in state leader and all others in state defeated. The election is initiated by any non-empty subset of nodes spontaneously starting execution of the algorithm. Each node has a distinct id from some totally ordered set $\mathcal{I}$. We assume asynchronous, point-to-point communication by means of message passing. Each message will be eventually delivered after finite, but unpredictable time. The complexity measure is the worst case message complexity, using messages of size $O(\log |D|)$.

The problem of leader election is one of the fundamental problems in distributed computing. Many distributed algorithms require one node to act as a central coordinator, manager of a common resource, or arbiter; hence the need for a leader. Even if the system starts with a coordinator, leader election is needed in order to achieve robustness (a new coordinator should be found after the crash of the previous one). In addition, several common problems (like spanning tree construction or finding the minimum/maximum of the values stored at the nodes) are equivalent to leader election, further stressing its importance.

The problem of leader election has been first considered in ring networks [12, 13]. Soon, a general algorithm for arbitrary networks has been developed [11]. As the message complexity of this algorithm is optimal, the subsequent research focused on how to exploit the structural information to improve upon [11]. The first wave of results assumed topological awareness and strong types of SD. In such setting, optimal linear algorithms have been achieved for several topologies, including complete graphs [14], some particular chordal rings [2], square tori [18] and hypercubes [9, 21]. Later, research shifted to relaxing requirements on structural information. In particular, linear or close to linear solutions have been achieved with topological awareness [4, 15] or topological awareness combined with weaker types of SD [6].

In this paper we make a further step in this direction, by assuming a very weak knowledge on the network topology. More precisely, we consider the class $\mathcal{C}G_4$ of Abelian Cayley graphs of degree 4, and we assume that the nodes only know that they are in a graph of $\mathcal{C}G_4$. They do not have SD, neither they know the size of the network. We show that even in this setting, a leader can be elected using $O(N + D\log(D^2/N))$ messages, where $N$ denotes the order and $D$ the diameter of the network. This bound is tight in the sense that there is a graph of $\mathcal{C}G_4$, namely the torus $d \times N/d$, $d \in \Theta(D)$, for which a matching lower bound is easily obtained from the $\Omega(d \log k)$ lower bound for ring of size $d$, with $k = d^2/N$ initial candidates: Assume there is an election algorithm in torus $d \times N/d$ ($d \geq N/d$, with
d in horizontal direction) of complexity $C$. Then the number of messages sent in the horizontal direction in the case there are $d^2/N$ candidates, each in different column, is $C' \leq C$. Collapsing the columns into single nodes means there is an algorithm for ring of size $d$, starting with $d^2/N$ candidates, of cost $C'$ messages.

The first linear algorithm for square tori\footnote{As shown earlier, linear election algorithm is possible only for tori with constant ratio of its sides.} has been given in [18]. The basic idea of this algorithm is that a leader candidate can exclusively claim a set of $O(l^2)$ nodes by spending only $O(l)$ messages. When $l$ reaches $\sqrt{N}$, there is only a constant number of candidates left and a leader is elected among them in a special terminal phase. It is noted in [20] that the same approach can be applied also to chordal rings with one chord of length $\Theta(\sqrt{N})$, with the only difference in the terminal phase. This is due to the fact that chordal rings and tori look locally the same. Again in square tori and chordal rings with one chord of length $\Theta(\sqrt{N})$, it is shown in [15] that SD is not necessary. The leader election can be solved by adding a preprocessing phase, called handrail technique, in which nodes locally identify opposite links.

Our algorithm applies for the class of graphs $C\mathcal{G}_4$, which includes tori of arbitrary sizes and chordal rings with one chord of arbitrary length. A nice feature of the graphs in $C\mathcal{G}_4$ is that we can represent the adjacencies as a figure in the plane, by drawing a square to represent a node, and by placing adjacent squares to represent adjacent nodes (see figure 1). With this representation, which is locally the same as the representation of the torus, a tessellation of the plane is obtained, where the nodes are repeated in the plane according to a 2-dimensional lattice. This geometrical approach has been successfully applied to compute the diameter [22], as well as to define routing schemes with good properties with respect to routing vulnerability [8], compact routings [16], and optical routings [17].

The main idea of the algorithm we propose is the reduction of the problem of election in a graph in $C\mathcal{G}_4$ to an election problem in the corresponding tessellation. Moreover, we show that the handrail technique [15] can be applied to most of the graphs of $C\mathcal{G}_4$, and it fails only in a small number of cases, where the diameter of the network is $\Omega(N)$. (In such cases, the algorithm in [11] for arbitrary networks can be used, because it gives the same message complexity.) As a consequence, our algorithm applies to all graphs in $C\mathcal{G}_4$, not requiring the knowledge of the network topology. In addition, the algorithm works uniformly, without the need of a terminal phase. It is worth mentioning that the synchronization technique of [5] is used, as it seems suitable for the analysis and further extensions of our algorithm to higher dimensions.

The structure of this paper is as follows. Section 2 contains the formal definitions of the graphs and the associated tessellation, as well as some properties of the corresponding lattice that we use later. In Section 3 we describe the algorithm, prove its correctness and give the analysis of its complexity. Section 4 concludes by pointing to possible future directions.
2 Definitions and preliminary results

The class of graphs we are interested in is defined as follows.

**Definition 1** $\mathcal{C}G_4$ is the class of undirected Abelian Cayley graphs of degree 4, i.e., the graphs $(\Gamma, S)$ with $\Gamma$ Abelian group of rank at most 2, and $S = \{\pm \alpha, \pm \beta\}$ a generating set of $\Gamma$.

Given $G = (\Gamma, S) \in \mathcal{C}G_4$, the set of nodes of $G$ is $\Gamma$, and every node $u$ is adjacent to the four nodes $u - \alpha$, $u + \alpha$, $u - \beta$, and $u + \beta$.

This class of graphs is the union of two classes: the class of circulant undirected graphs of degree 4, and the class of 2-dimensional torus. Note that the intersection of these two classes is not empty, since if $\gcd(m, n) = 1$, then $\mathbb{Z}_m \times \mathbb{Z}_n$ is isomorphic to $\mathbb{Z}_{mn}$. Note also that the graphs known as chordal rings are those graphs in $\mathcal{C}G_4$ with one of the generators equal to 1.

We can also define $\mathcal{C}G_4$ as the class of graphs that can be represented by a periodic tessellation of the plane with square unitary cells. In order to define and give some properties of this geometrical representation of the graphs in $\mathcal{C}G_4$, we identify the group $\mathbb{Z}^2 = \{(x, y), x, y \in \mathbb{Z}\}$ with the set of unitary squares with center in $\mathbb{Z}^2$. In this way, the square $(x, y)$ is the square of side 1 with center in $(x, y)$, and this set of squares constitutes a regular infinite tessellation of the plane.

Let $G = (\Gamma, \{\pm \alpha, \pm \beta\})$ be a graph in $\mathcal{C}G_4$ of order $N$. We can associate a node of $G$ to every square of $\mathbb{Z}^2$, so that adjacent nodes correspond to adjacent squares, by associating the first generator, $\pm \alpha$, to the horizontal direction, and the second generator, $\pm \beta$, to the vertical direction. Assuming that the square $(0, 0)$ contains the node $0$ of the group $\Gamma$, then the square $(x, y)$ contains the node $x \cdot \alpha + y \cdot \beta$ of the group $\Gamma$. By following this rule we obtain a labelling of the plane with all the nodes of the graph periodically repeated. A succession of adjacent squares in the tessellation can be viewed either as a succession of generators, that is, a path of the graph, or as a vector in the plane. The distance between two nodes $u, v \in G$ is the length, in norm $L_1$ (or Manhattan distance), of the smallest vector $u - v$. 

![Figure 1: The geometrical representation of $(\mathbb{Z}_{14}, \{\pm 2, \pm 3\})$](image-url)
We say that a basis \( CU \) and only if \( \det \) is a special \( B4 \) if \( \label \) \( L \) and only if they differ by a vector of \( (\mathbb{Z}) \). Property 2: If the set of two vectors \( \{v\} \) determined by \( L \) and \( v \) \( B4 \) satisfies \( x \cdot \alpha + y \cdot \beta = 0 \) in the group \( \Gamma \). That is

\[
(x, y) \in \mathcal{L}(G) \Leftrightarrow \begin{cases} x \cdot \alpha + y \cdot \beta \equiv 0 \mod N & \text{if } \Gamma = \mathbb{Z}_N \\ x = \lambda_1 \cdot m \text{ and } y = \lambda_2 \cdot n & \text{if } \Gamma = \mathbb{Z}_m \times \mathbb{Z}_n \end{cases}
\]

Actually, the labelling \( v_G \) is a group epimorphism from \( \mathbb{Z}^2 \) into \( \Gamma \), and the lattice \( \mathcal{L}(G) \) is the kernel of \( v_G \). This implies that two squares contain the same node if and only if they differ by a vector of \( \mathcal{L}(G) \), i.e., \( v_G(x_1, y_1) = v_G(x_2, y_2) \) if and only if \( (x_2 - x_1, y_2 - y_1) \in \mathcal{L}(G) \).

**Definition 3** A basis of \( \mathcal{L}(G) \) is a set \( \{v_1, v_2\} \) of two generating vectors of \( \mathcal{L}(G) \). We say that a basis \( \{v_1, v_2\} \) is packed if and only if every vector \( w \in CG_4 \) satisfies \( |w| \geq |v_i| \), for \( i = 1, 2 \).

With these definitions, we have the following properties.

**Property 1** ([7]) A set of two vectors \( \{v_1, v_2\} \subset \mathcal{L}(G) \) form a basis of \( \mathcal{L}(G) \) if and only if \( \det(v_1, v_2) = N \).

**Property 2** ([22]) If the set of two vectors \( \{v_1, v_2\} \) form a packed basis of \( \mathcal{L}(G) \) then the diameter of \( G \) is \( D = \min\{\frac{|v_1 - v_2|}{2}, \frac{|v_1 + v_2|}{2}\} \).

In fact, the vectors \( v_1 - v_2 \) and \( v_1 + v_2 \) are the diagonals of the parallelogram determined by \( v_1 \) and \( v_2 \). This result is based on the fact that the parallelogram determined by the two vectors of the basis contains all the nodes of the graph exactly once (for more details on the properties of the packed basis see [16, 22]).

**Definition 5** A packed basis \( \{v_1, v_2\} \) of \( \mathcal{L}(G) \), with \( v_1 = (a_1, b_1) \) and \( v_2 = (a_2, b_2) \), is a special basis if the three following conditions are satisfied.

1. \( |v_1| \geq |v_2| \), i.e., the shortest vector is \( v_2 \).
2. \( a_1 > b_1 \geq 0 \), i.e., \( v_1 \) is in the first quadrant with larger first coordinate.
3. \( b_2 \geq b_1 \), i.e., the end-point of \( v_2 \) is in the region above the line \( y = b_1 \).
**Observation 1** If $G$ and $G'$ are two graphs in $\mathcal{G}_4$ then it can be proved that, $G \cong G'$ if and only if the two lattices $\mathcal{L}(G)$ and $\mathcal{L}(G')$ either are the same or are symmetric.

Thanks to Observation 1, we can assume, w.l.o.g., that there is a special basis of $\mathcal{L}(G)$. The relative position of the two vectors of a special basis given by the following lemma.

**Lemma 1** Let $v_1 = (a_1, b_1)$ and $v_2 = (a_2, b_2)$ be the two vectors of a special basis of $\mathcal{L}(G)$. Then

$$0 \leq a_2 + b_2 \leq a_1 + b_1, \quad 2b_1 \leq -a_2 + b_2 \leq a_1 + b_1, \quad \text{and} \quad b_1 \leq b_2$$

**Proof.** The third inequality is exactly the condition 3 of Definition 5. The diagonals of the parallelogram determined by $v_1$ and $v_2$ are longer than $v_1$ and $v_2$, because they belong to $\mathcal{L}(G)$ and the vectors of the basis are taken as short as possible in $\mathcal{L}(G)$. This, together with the condition 2 of Definition 5, implies

1. $v_2 \in B(0, \|v_1\|)$,
2. $v_2 \not\in B(v_1, 1 - \|v_1\|)$, and
3. $v_2 \not\in B(\mathbb{Z}^2, 1 - \|v_1\|)$

where $B(c, r)$ denotes the ball of center $c$ and radius $r$ in norm $L_1$. This is equivalent to say that $v_2$ belongs to the plane region limited by the lines $y = b_1$, $y - x = \|v_1\|$, $y - x = 2b_1$, $x + y = 0$, and $x + y = \|v_1\|$, which gives the inequalities in the statement of the lemma.

The following corollary is a consequence of the previous lemma. It will be useful in the proof of lemmas 2 and 3.

**Corollary 1** Let $v_1 = (a_1, b_1)$ and $v_2 = (a_2, b_2)$ be the two vectors of a special basis of $\mathcal{L}(G)$.

1. If $a_2 \geq 0$ then $a_1 \geq 2a_2$ and $b_2 \geq 2b_1$.
2. If $a_2 < 0$ then $|a_2| \leq a_1$.

**Proof.** From Lemma 1, we have $a_2 + b_2 \leq a_1 + b_1$ and $2b_1 \leq -a_2 + b_2$. By adding these two inequalities, we get $2a_2 \leq a_1 - b_1$ and thus, $2a_2 \leq a_1$. On the other hand, if $a_2 \geq 0$, then $b_2 \geq -a_2 + b_2 \geq 2b_1$. This completes the proof of (1).

If $a_2 < 0$ then the inequality $|a_2| \leq a_1$ is a direct consequence of the inequalities $-a_2 + b_2 \leq a_1 + b_1$ and $b_1 \leq b_2$ of Lemma 1.

The following lemmas will be used in the analysis of our leader election algorithm. In all of them $v_1 = (a_1, b_1)$ and $v_2 = (a_2, b_2)$ denote the two vectors of a special basis of $\mathcal{L}(G)$, and for every $\mathcal{A} \subseteq \mathbb{Z}_2^2$, $|\mathcal{A}|$ and $\#\mathcal{A}$ denote the number of squares of $\mathbb{Z}_2^2$ and the number of nodes of $G$ contained in $\mathcal{A}$, respectively. (Note that $\#\mathcal{A}$ is the cardinality of the set $v_G(\mathcal{A})$.) The lemmas correspond to the
three cases: (1) \( \mathbf{v}_2 \) is in the first quadrant, (2) \( \mathbf{v}_2 \) is in the second quadrant and \( D = \frac{a_2^2}{2a_1} \), and (3) \( \mathbf{v}_2 \) is in the second quadrant, and \( D = \frac{\sqrt{a_2^2 + a_1^2}}{2} \). These three cases are represented in Figure 2. Since the proof of these lemmas is similar, we only give the proof of Lemma 2.

**Lemma 2** If \( a_2 \geq 0 \), \( \mathcal{R} = [0,a_1] \times [0,b_1] \), and \( S \) is a square of size \( L \) contained in \( \mathcal{R} \), then \( \# \mathcal{R} = N, \max\{a_1,b_1\} \in \Theta(D) \), and \( \min\{a_1,b_1\} \in \Theta(N/D) \).

**Proof.** The rectangle \( [a_1-a_2,a_1] \times [0,b_1] \) is the translation of the rectangle \( [0,a_2] \times [b_2-b_1,b_2] \) by the vector \( \mathbf{v}_1 - \mathbf{v}_2 \in \mathcal{L}(G) \), and thus both rectangles contain exactly the same nodes. Moreover, one can see that, if \( (x,y) \in \mathcal{R} \setminus \{0,a_2\} \times \{b_2-b_1,b_2\} \), then \( (x,y) + \mathbf{v}_1 - \mathbf{v}_2 \) is not in \( \mathcal{R} \). Let us show that there are no more repeated nodes. We need only to prove that if two squares in \( \mathcal{R} \) contain the same node, then they differ by the vector \( \mathbf{v}_1 - \mathbf{v}_2 \).

Let \( (x_1,y_1) \) and \( (x_2,y_2) \) be two squares in \( \mathcal{R} \) satisfying \( \mathcal{L}(x_1,y_1) = \mathcal{L}(x_2,y_2) \). Then the vector \( \mathbf{w} = (w_1, w_2) \), with \( w_1 = x_2 - x_1 \) and \( w_2 = y_2 - y_1 \), belongs to \( \mathcal{L}(G) \) and satisfies \( |w_1| \leq a_1 \) and \( |w_2| \leq b_2 \). But the vector \( \mathbf{w} \) cannot be in \( \mathcal{R} \), since \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) form a basis of \( \mathcal{L}(G) \). Therefore, one of the two coordinates of \( \mathbf{w} \) is positive, and the other one is negative. Assume, w.l.o.g., that \( 0 < w_1 < a_1 \) and \( 0 < -w_2 < b_2 \). The two vectors \( \mathbf{v}_1 - \mathbf{w} \) and \( \mathbf{v}_2 + \mathbf{w} \) are in \( \mathcal{L}(G) \), thus they cannot be in \( \mathcal{R} \). This implies \( a_1 - a_2 \leq w_1 \) and \( b_2 - b_1 \leq -w_2 \). Now we have that \( a_1 - a_2 \leq w_1 < a_1 \) and \( b_2 - b_1 \leq -w_2 < b_2 \). Let us consider the vector \( \mathbf{w} + \mathbf{v}_2 - \mathbf{v}_1 \), which is in \( \mathcal{L}(G) \). From the above inequalities, \( 0 \leq |\mathbf{w} + \mathbf{v}_2 - \mathbf{v}_1| < a_2 + b_1 \leq |\mathbf{v}_2| \). Since \( \mathbf{v}_2 \) is a shortest vector in \( \mathcal{L}(G) \), we have \( \mathbf{w} = \mathbf{v}_1 - \mathbf{v}_2 \). This implies \( \# \mathcal{R} = N \), because \( \mathcal{R} \) contains \( a_1 b_2 \) squares, with exactly \( a_2 b_1 \) repeated nodes, that is, \( \# \mathcal{R} = a_1 b_2 - a_2 b_1 = \det(\mathbf{v}_1, \mathbf{v}_2) = N \), by Property 1.

The shortest diagonal of the parallelogram determined by \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) is \( \mathbf{v}_1 - \mathbf{v}_2 \), which is no shorter than \( \mathbf{v}_1 \). It follows from Property 2 that \( D = \frac{1}{2}(a_1-a_2+b_2-b_1) \leq \frac{1}{2}(a_1+b_2) \leq \max\{a_1,b_2\} \leq \max\{a_1+b_1,a_2+b_2\} = |\mathbf{v}_1| \leq 2D \). Therefore, \( \max\{a_1,b_2\} \in \Theta(D) \). Since \( N = a_1 b_2 - a_2 b_1 \), and \( a_2 b_1 \leq \frac{1}{4} a_1 b_2 \) (from Corol-
lary 1), we have also that \( \frac{3}{4} a_1 b_2 \leq N \leq a_1 b_2 \), that is, \( a_1 b_2 \in \Theta(N) \). Hence, we can derive that \( \min\{a_1, b_2\} \in \Theta(N/D) \), which concludes the proof. \( \square \)

**Lemma 3** If \( a_2 < 0, |a_2| \leq b_1 \), \( R = [a_2, a_1] \times [0, b_2] \) and \( S \) is a square of size \( \ell \) contained in \( R \), then \( \#R = N, \max\{a_1 + |a_2|, b_2\} \in \Theta(D) \), and \( \min\{a_1 + |a_2|, b_2\} \in \Theta(N/D) \).

**Lemma 4** If \( a_2 < 0, |a_2| > b_1 \), \( R = [0, a_1] \times [0, b_1 + b_2] \), and \( S \) is a square of size \( \ell \) contained in \( R \), then \( \#R = N, \max\{a_1, b_1 + b_2\} \in \Theta(D) \), and \( \min\{a_1, b_1 + b_2\} \in \Theta(N/D) \).

The next corollary is used in Section 3, in the analysis of the complexity of our leader election algorithm.

**Corollary 2** There is a rectangle \( R \) of size \( x \times y \) such that \( \#R = N, D \leq x < cD \) for some constant \( c \) and \( y \in \Theta(N/D) \).

**Proof.** The corollary follows directly from the fact that any special basis of \( L(G) \) satisfies the hypothesis in the statement of one of the lemmas 2, 3 or 4. \( \square \)

Note that the rectangle \( R \) in the previous corollary satisfies \( |R| \in \Theta(N) \).

### 3 The leader election algorithm

Now, we are ready to describe our leader election algorithm. The intuition behind our algorithm is to reduce the problem of election in an Abelian Cayley graph of degree 4, \( G \), into the problem of election in a torus containing all the nodes of \( G \) (possibly with some repetitions) and of roughly the same diameter. This is accomplished by using the geometrical representation of the graph. In fact, we elect a leader of the rectangle \( R \) from the Corollary 2. Since \( R \) contains all nodes of \( G \), the resulting leader is also the leader of \( G \). As the wrap-around properties
of $G$ could be very different from those of the torus corresponding to $R$, the algorithm must be more robust, and a careful analysis is needed.

**Notation.** Let $s$ be a positive integer. The cross of $v$ at distance $s$ is the set $C_v(s) = \{v + x\alpha | -s < x < s\} \cup \{v + x\beta | -s < x < s\}$. The territory of $v$ of side $s$ is the set $T_v(s) = \{v + x\alpha + y\beta | 0 \leq x, y < s\}$. The sequence of distances $s_1, s_2, \ldots$ is chosen as $s_i = \gamma i$ for some constant $\gamma \geq 1$. We denote the corresponding crosses and territories by $C_v(s_i)$ and $T_v(s_i)$.

The algorithm proceeds in logical rounds. A spontaneously awaken node starts as a leader candidate in round one, a node awaken by an incoming message is automatically defeated and does not run for leadership. A node $v$ in round $i$ marks a cross $C_v$. If this cross intersects with a cross of a stronger candidate (in higher round, or in the same round, but with higher id), $v$ is defeated. Otherwise $v$ is promoted to the next round. When a candidate detects that its cross has “wrapped around” in both directions, it knows it is the leader. The efficiency of this approach is based on the exponential growing of the distances $s_i$ and the fact that by marking a cross of length $s_i$, and thus spending $O(s_i)$ messages, the candidate $v$ can exclusively claim a territory $T_v$, which can be of size $O(s_i^2)$ (see Figure 3 (a)).

In order to employ this technique, two problems have to be solved. The first problem arises due the lack of SD. Although no globally consistent labelling of the ports is available, a node must be able to forward a message via the port opposite to the port it arrived from, in order to mark the cross. This problem is solved by including in our algorithm a preprocessing phase from [15]. Unlike the case of square tori in [15], this preprocessing can fail in our setting. In such a case the algorithm for arbitrary networks from [11] is applied. We show that all of this does not inversely influence the complexity of our algorithm.

The second problem is related with the synchronization between rounds. Consider the situation in Figure 3. $C_v \cap C_u \neq \emptyset$, yet it is possible that both $u$ and $v$ proceed to the round $i + 1$. Let us assume that the node $v$, with lower id, marks its cross, while $u$ is very slow. Since $v$ has not seen anybody stronger, it proceeds to the next round. When $u$ marks its cross, it sees only weaker candidates and also proceeds to the next round.

In this situation, at the moment when $u$ marks its cross (and crosses $v$’s cross), $v$ is already in the next round, but $u$ is not aware of that. The main idea of the solution is to let $v$ announce in its cross if it died, and let $u$ wait for the status of $v$. If $v$ died, $u$ can proceed, if $v$ survived to the next round, $u$ becomes weaker and dies. Note that $u$ waits for $v$ only if $v$ is weaker than $u$, but it would be stronger in the case it reached the next round. We say that $v$ is *incomparable* with $u$ in such a case.

Next we give a detailed description of our leader election algorithm.
3.1 Algorithm ELECT

The algorithm works in two phases, the preprocessing phase and the promotion phase. Every node \( v \) does the following.

**Preprocessing phase.** The handrail technique of [15] is used to identify the pairs of opposite ports. One of them is called horizontal and the other one vertical. (Note that, in general, two different nodes do not agree on the horizontal and vertical directions. However, this agreement is not needed, since they only need to identify opposite ports.)

- If the identification can not be done, then \( v \) proceeds to the election by using the algorithm of [11]. (Note that, since the network is symmetric, if \( v \) can identify opposite ports, then every other node can identify opposite ports as well.)
- Otherwise, \( v \) enters the promotion phase.

**Promotion phase.** Once the preprocessing phase has been executed and the opposite links have been identified, \( v \) starts a sequence of rounds until it becomes leader or it dies. In round \( i \) it marks the cross \( C^*_v \) using 4 tokens, one for each part of the cross, and checks whether it collided with a cross of a stronger candidate.

1. **Token structure.** Each token carries the id of its owner, the round number \( i \), the counter count which maintains its distance from its owner and the direction dir (from \{horizontal, vertical\}) it travels, according to its owner. In addition, each token has a status field status, taking values from \{Mark, Die!, Wrap, OK, Dead\}. When we say “send token XX”, we mean sending a token with status XX, and with all the other fields set as well. The tokens \( p \) and \( q \) are compared according to its id, if they have the same round number, and according to its round number, otherwise.

2. **Local storage.** Each node \( v \) keeps a copy of the latest token from the strongest candidate it has seen so far (in a variable max), as well as field dirs, which maintains a set of directions from which the tokens of the strongest candidate have been received. In addition, a variable wait contains an incomparable token for which max’s owner needs acknowledgement whether it survived the current round or not. wait is set to the current value of max whenever a token \( m \) with status Mark arrives to \( v \) with \( m \geq \max, \max.status \in \{Wrap, OK\} \) and \( \max' > m \), where \( \max' \) equals \( \max \) with the round number incremented. The variable wait is cleared whenever a token \( m \) with \( m.id = \text{wait.id} \) and \( m.status = \text{Dead} \) is received. In addition, wait is cleared whenever a token \( m \) with \( m > \text{wait'} \) arrives to \( v \).

All these variables are updated whenever a token \( m \) arrives to \( v \), but after \( m \) has been processed; wait is updated before the update of max.
Actions performed by each candidate during round $i, i \geq 1$.

- A token performing Mark&Check is sent over each of the four ports.
- $v$ waits for the return of all its tokens. Then it performs Evaluation.

Mark&Check. Each token $m$ marks and checks one part of the cross. It starts with status Mark and travels straight in the direction it left $v$ until one of the following conditions is fulfilled (checked in this order).

- A node visited by a stronger candidate is found ($m < \text{max}$). In this case $m$ returns back to $v$ with status set to Die!.
- A wraparound is detected ($m.id = \text{max}.id$ and $m.dir \in \text{dirs}$). In this case $m$ is sent back to $v$ with status Wrap.
- The distance $s_i$ has been travelled ($m.count$ has reached $s_i$). In this case $m$ is sent back to $v$ with status OK.

A token with status Die! returns directly to its owner without further checks. However, whenever during its return to $v$ a token $m$ with status Wrap or OK arrives to a node (but also in the node in which it turns back), the following checks (in this order) are executed:

- If wait is set, $m$ waits until wait is cleared.
- If $m < \text{max}$, the status of $m$ is set to Die!.

After performing these tests, $m$ continues its travel towards $v$.

Evaluation. Once all the tokens have come back to $v$, $v$ decides the further action. There are three possible cases.

- If at least one of the tokens has status Die; $v$ is defeated and does not proceed to the next round. The death of $v$ is broadcasted by having each token travel the part of the cross it traversed in the Mark&Check phase, now with status Dead.
- If all tokens returned with status Wrap, the algorithm ELECT terminates and $v$ is declared leader. The identity of leader is then broadcasted using flooding (at the cost of $3N$ messages).
- Otherwise $v$ successfully claims $T_v'$ and proceeds to the next round.

3.2 Analysis

In this section we prove the correctness of the algorithm ELECT, as well as analyze its message complexity.
3.2.1 Correctness

Lemma 5 Every candidate receives back all its tokens, or it learns that the leader has been elected.

Proof. Assume the lemma does not hold. Let \( v \) be the weakest candidate such that it has not received back all of its tokens after Mark\&Check. This means that at least one of its tokens is waiting for a token from an incomparable (and hence weaker) candidate \( u \). If \( u \) has received all its tokens and progressed to evaluation, this token would have been awakened, or \( v \) would have learned that \( u \) is the leader. Since neither happened, \( u \) has not received its token back, which gives a contradiction because \( u \) is weaker than \( v \).

Lemma 6 If \( C^i_v \cap C^i_u \) then at most one of \( u \) and \( v \) survives round \( i \).

Proof. Suppose both \( u \) and \( v \) survive to the next round. Let \( id_u < id_v \). If there is a node \( w \in C^i_v \cap C^i_u \) such that \( w \) has been marked by \( v \) before being reached by \( u \), then \( u \) dies, because one of its tokens will come to a node already visited by a stronger candidate. If all nodes of \( C^i_v \cap C^i_u \) have been marked by \( u \) before being reached by \( v \), then \( v \) dies, because it will find \( u \) incomparable and wait for acknowledgement that \( u \) has died. Instead, the tokens of \( u \) in round \( i + 1 \) will come and kill it.

Theorem 1 The algorithm ELECT correctly elects a leader in any Abelian Cayley graph of degree 4.

Proof. If the handrail technique fails, the algorithm elects a leader. Otherwise, the theorem follows from the following facts:

1. At any round, at least one candidate remains active. The strongest candidate \( v \) can die only when it waits for confirmation of death of an incomparable candidate \( u \). If \( v \) dies, \( u \) has survived.

2. At the end of the algorithm, at most one candidate is declared leader. Only a candidate \( v \) to which all its tokens returned in status Wrap is declared leader. However, this means \( v \) has marked a complete vertical and horizontal line. It follows from Lemma 6 that there could not be two such candidates.

3. The algorithm eventually terminates. Lemma 5 makes sure that the algorithm progresses to higher rounds. Since each line segment of at least \( N \) nodes contains repetitions (and hence wrap-around), the leader will be declared, at least when \( s_i \) exceeds \( N \). This will eventually occur, because the sequence \( s_i \) is strictly increasing.
3.2.2 Complexity

The message complexity of the algorithm \texttt{ELECT} is the maximum of the complexities of the two phases, that is, (1) the preprocessing and the promotion phases, for graphs in which preprocessing succeeds, and (2) the preprocessing phase and the election algorithm in [11], for graphs in which preprocessing fails.

It follows from [15] that the preprocessing phase takes $O(N)$ messages, even if the pairs of opposite ports can not be determined, (that is, if it fails). This result comes from the fact that, the handrail technique needs 12 messages per node, and it succeeds exactly when the following property holds.

\textbf{Property 3} Two neighbors of a node $v$ correspond to opposite ports if and only if their only common neighbor is $v$.

\textbf{Lemma 7} Let $G \in \mathcal{C}G_d$. If the handrail technique fails then the diameter of $G$ is in $\Omega(N)$.

\textbf{Proof.} (Sketch.) If $v_G(x,y) = v$, then the set $\mathcal{A}$ of squares at distance at most 2 of $(x,y)$ contains $|\mathcal{A}| = 13$ squares. Assume that Property 3 is not satisfied. Then $\#\mathcal{A} < 13$, that is, the set of squares at distance at most 2 contains some repeated node. This implies that there is a vector in $\mathcal{L}(G)$ with norm at most 4, and thus, in any special basis, $|v_2| \leq 4$.

From properties 1 and 2, we can conclude that, the diameter of $G$ is in $\Omega(N)$.

The analysis of the promotion phase of algorithm \texttt{ELECT} is based on bounding the number of candidates in each round, by using the fact that in the previous round each of them exclusively claimed large territory. In order to do that, we need the following lemma.

\textbf{Lemma 8} If $C'_v \cap C'_u = \emptyset$ then $T'_v \cap T'_u = \emptyset$.

\textbf{Proof.} Suppose $T'_v \cap T'_u \neq \emptyset$. We need only to prove that $C'_v \cap C'_u \neq \emptyset$.

Let $w \in T'_v \cap T'_u$, i.e., $w = v + x, \alpha + y, \beta = u + x, \alpha + y, \beta$, for some $0 \leq x_u, x_v, y_u, y_v < s_i$. The two integers $x'$ and $y'$ defined by $x' = x_v - x_u$ and $y' = y_v - y_u$ satisfy $-s_i < x' < s_i$ and $-s_i < y' < s_i$. Let us consider now the node $w' = w - x, \alpha - y, \beta$. One can easily check that $w' = v + y, \beta$ and $w' = u - x, \alpha$, which imply that $w' \in C'_v \cap C'_u$.

As a consequence of Lemma 8, if $u$ has successfully marked $C'_u$, it can exclusively claim $T'_u$.

\textbf{Theorem 2} The communication complexity of the algorithm \texttt{ELECT} is $O(N + D \log(D^2/N))$.

\textbf{Proof.} If preprocessing fails then the communication complexity is determined by the complexity of election using the algorithm in [11], i.e., $O(N \log N + |E|)$. 

As $|E| = 2N$ and $D = \Omega(N)$ (due to Lemma 7), the communication complexity can also be expressed as $O(N + D\log(D^2/N))$.

Let us assume now that the preprocessing succeeded. We only need to show that the communication complexity of the promotion phase is $O(N + D\log(D^2/N))$. From Corollary 2 we know that there is a rectangle $R$ of sides $d \times m$ that contains all nodes of the graph, $D < d < cD$ for some constant $c$ and $m \in \Theta(N/D)$. Clearly, $md = n \in \Theta(N)$.

Let $C_t$ be the number of candidates which survived round $i$. As the cost of marking a cross in round $i$ is clearly $O(s_t)$, the total complexity can be estimated as $\sum_{i=1}^t O(s_t)C_{t-1}$ where $t$ is the round in which the leader was determined, and $C_0$ is at most $N$. The estimate of $C_t$ is divided into three cases:

**Case $s_t \leq m$.** As $|T^i_c| = s_t^2$, due to Lemma 8 the claimed areas of at most $n/s_t^2$ candidates can fully fit into $R$. In addition there are at most $1 + (d + m)/s_t$ candidates whose areas are only partially within $R$, hence $C_t$ is bounded by $O(n/s_t^2)$.

**Case $m < s_t < d$.** Since the area of each candidate (except the rightmost) exclusively takes a segment of length $s_t$ from the upper border of $R$, the $C_t$ is bounded by $1 + d/s_t \in O(n/ms_t)$.

**Case $d \leq s_t$.** $T^i_c$ wraps in both directions and covers the whole $R$, hence $C_t = 1$.

Since $s_t = \gamma$ for some constant $\gamma > 1, n \in \Theta(N), d \in \Theta(D)$ and $m \in \Theta(N/D)$, taking the sum of the three cases above yields

$$\sum_{s_t \leq N} O(s_{t+1}C_t) = \sum_{s_t \leq m} O(s_{t+1}n/s_t) + \sum_{m \leq s_t < d} O(s_{t+1}n/ms_t) + \sum_{d \leq s_t} O(s_t) =$$

$$= \sum_{m \leq s_t \leq d} O(n/\gamma) + \sum_{m \leq s_t < d} O(n/m) + \sum_{d \leq s_t} O(\gamma) = O(n) + O(n/m)\log(d/m) + O(N) = O(N + D\log(D^2/N)) \quad \square$$

### 4 Conclusions

In this paper, we have presented an $O(N + D\log(D^2/N))$ election algorithm for Abelian Cayley graphs of degree 4. The key property of our algorithm is the fact that it does not require the knowledge of the actual topology, its size, or any kind of sense of direction. Although it is a nice generalization of the algorithms from [15], we feel the main contribution of our paper is the technique allowing to reduce the problem of election in an Abelian Cayley graph to the study of properties of the corresponding tessellation.

We consider this paper a work in progress, with the main goal to extend the results also to Abelian Cayley graphs with higher degree and study the impact of structural information on them. In particular, we are interested in the following three models: (1) No topological awareness, without any SD. (2) No topological awareness, with orientation (ports corresponding to the same generator labelled
by the same symbol) and (3) Topological awareness with orientation (i.e. knowing the underlying group and the generators, ports labelled by corresponding generators). For each case we would like to identify the widest class of networks such that optimal (i.e. $O(N + D \log(D^2/N))$) election is possible. This is not easy undertaking, as there are serious obstacles on the road towards higher degrees:

- The cost of the preprocessing phase is $O(\delta^2)$ where $\delta$ is the degree of the graph, limiting our hope in the case (1) to graphs of constant degree.
- Lemma 7 does not hold in higher dimensions, pointing to a need of more elaborate handling of the case when preprocessing fails.
- Observation 1 allowed us to reduce the study of $L(G)$ to lattices with specific basis. The result is a consequence of the equivalence between Ádám-isomorphism and isomorphism in the 2-dimensional case (see [1, 3]). This is no longer true in the 3 and more dimensions ($\delta \geq 6$).
- A generalization of Corollary 2 to higher dimensions is needed to support analysis.

References


Lali Barrière is with the Departament de Matemàtica Aplicada i Telemàtica, Univ. Politècnica de Catalunya, Barcelona, E-mail: lali@mat.upc.es.

Stefan Dobrev is with the Department of Informatics, Institute for Mathematics, Slovak Academy of Sciences, Bratislava, E-mail: stefan@ifi.savba.sk.
Cycle Covering

J-C. Bermond  
CNRS-I3S-INRIA, France

L. Chacon  
France Telecom R&D, France

D. Coudert  
CNRS-I3S-INRIA, France

F. Tillerot  
France Telecom R&D, France

Abstract

This paper considers the design of a survivable WDM network based on covering the initial network with sub-networks, which are protected independently from each other. We focus on the case where the optical WDM network is a ring, there are requests between any pair of vertices and the covering is done with small cycles. This problem can be modelled as follows: Find a covering of the edges of a logical graph $I$ (here the complete graph $K_n$) by subgraphs $I_k$ of a certain kind (here cycles $C_k$ of length $k$), such that, for each $I_k$, there exists in the physical graph $G$ (here $C_n$) a disjoint routing of the edges of $I_k$. The aim is to minimize the number of subgraphs $I_k$ in the covering. We give optimal solutions for that problem.

Keywords

cycle covering, WDM, graph, survivability

1 Introduction

With the growth in data traffic and the surging demand for new services, higher capacity core networks are needed. Wavelength Division Multiplexing (WDM) technology is expected to provide solutions to this challenge. Early deployment of WDM technology has been point-to-point to increase link bandwidth by carrying several high bit-rate signals on different wavelengths in a same fiber (optical channels). Beyond this fiber capacity expansion, recent advances in WDM equipment focus on optical networking, with the ability to add, drop and construct wavelength routed networks. Routing and resource allocation problems in
non survivable WDM networks have been studied for several years [1, 2, 8, 14]. Network survivability [17] (i.e. the ability to recover traffic affected by failures) is becoming a key issue in the design of ultra-high capacity networks based on WDM technology. This study considers the design of a survivable WDM network based on covering the initial network with sub-networks, which are protected independently from each other.

The planning of an optical layer can be divided into two sub-problems: computing the routing of the wavelength demands on the physical layer (routing problem) and allocating the resources to the routed demands (resource allocation problem) [1, 8, 14]. The survivability against equipment or link failure consists in computing new routes for the demands affected by a failure; thus the optical layer must be over-dimensioned. Two survivability schemes can be implemented: protection or restoration. Protection can be done by using a pre-assigned capacity between nodes in order to replace the failed or degraded transport entities. On the other hand, restoration can be realized by using any capacity available between nodes in order to find a transport entity that can replace the failed one. Furthermore, restoration is based on re-routing algorithms to find a new path to recover failed network entities, at the time the failure occurs. Dividing the network into independent sub-networks provides an intermediate solution for survivability. Indeed it allows resource sharing within the limits of a given sub-network, and uses of fast automatic protection in case of failure [16].

This paper focuses on the design of an optical layer based on survivable subnetworks. The objective is to cover the initial set of demands (logical graph) by a set of cycles. The ring topology is chosen as sub-network since it minimizes the complexity of the routing problem with full survivability for any single failure. Indeed we used on the cycle half of the capacity for the demands and in case of failure we reroute the traffic going through the failed link via the remaining part of the cycle using the other half of capacity. It will be interesting to get very small cycles as subnetworks as they are more easy to manage and less costly to reroute. Also, we will associate a wavelength to each cycle (in fact two: one for the normal traffic and one for the spare one). Furthermore this cycle should satisfy the disjoint routing cycle (DRC) property, implying that it is embedded in an elementary cycle of the physical graph.

Our final aim is to minimize the cost function of the network. This cost function depends on many parameters and is difficult to study in full generality; in first approximation, we can reduce it to minimize the number of cycles of the covering.

Finally, we will restrict ourselves to a particular case of the general problem which is interesting for practical networks and shows already the difficulty of the problem. Mainly, we will suppose that the physical network is a ring and the set of requests is the All-to-All one.

Compared to previous studies based on a similar approach [10, 6], we provide an analytical model and optimal solutions for the ring physical topology.
A similar problem is considered in [5] and [7]. They use the same ring survivability conditions but their aim is to minimize the sum of the number of vertices of the rings.

In the next section, we precise our model and hypotheses. In section 3, we recall related results from design theory, and in section 4, we give the solution for a covering with the minimum number of cycles showing that such a solution is attained with cycles of length at most 4 which is very good for our original problem.

2 Modelling

We model the optical telecommunications network by a graph, called the physical graph and denoted by $G$. The vertices of the graph represent the optical switches and the edges the fiber-optics links. In fact, $G$ is an oriented symmetric multigraph; indeed each time there is a fiber optic from a node $x$ to a node $y$ there is also the opposite one and if there are $k$ such fibers we should consider in the model $k$ arcs from $x$ to $y$, and $k$ arcs from $y$ to $x$. For simplicity we consider simple graphs and, for reasons stated after, we can consider instead of the symmetric digraph the underlying undirected graph obtained by replacing each pair of symmetric arcs by an edge.

Usually the physical graph has no regularity properties, just enough connectivity in order to insure a good routing for the demands even in case of failures. However many designers of an optical network builds it with loops interconnected between themselves. Thus, the first case to consider is when the physical graph is a loop (or ring), which means that the graph $G$ is a cycle (ring). So, in the following, we will suppose that the graph $G$ is an undirected cycle of length $n$, denoted $C_n$.

The family of requests (or demands), called instance of communications, is modelled by a graph, called logical (or virtual) graph and denoted by $I$. The vertices represent the nodes of the physical graph and the edges correspond to the requests between these nodes. In general, it is a digraph but in telecommunication networks used for applications like telephone the requests are symmetric (i.e. for each communication request from node $A$ to node $B$ there is a similar request from node $B$ to node $A$). Thus, the logical graph will be undirected. Furthermore, we will suppose that the routing of symmetric requests is done by a symmetric routing. There is no constraint forcing that, but it is the way requests are actually managed. So the symmetry of the routing explains why we consider undirected graphs for the physical one. Finally the instance of communication called total exchange (or All-to-All), where each node wants to communicate with all the others simultaneously, is important. In such a case, the logical graph $I$ will be the complete graph. In the following sections, we will suppose that the logical graph $I$ is the complete graph on $n$ vertices, $K_n$. 
Routing an instance consists in finding a set of paths such that to each (sym-metric) request (edge of \( I \)) is associated a path in the physical graph \( G \) between the pairs of nodes communicating in the request. Here we do not consider the allocation of wavelengths to the request (that is done later in the last phase of the network design). The securization problem considered in the introduction can be modelled by finding a covering of the edges of the logical graph \( I \) by subgraphs \( I_k \). In general, we wish the \( I_k \) to have a simple structure and a small number of vertices. Therefore the interesting case is when \( I_k \) is a small cycle. Indeed, on the cycle we use half of the capacity for the demands, and in case of failure we reroute the traffic through the failed link via the remaining part of the cycle using the other half of the capacity. It will be interesting to get very small cycles as subnetworks as they are easier to manage and less costly to reroute. Also, we will associate a wavelength to each cycle (in fact two: one for the normal traffic and one for the spare one). The problem of finding a covering of the edges of a graph (in particular \( K_n \)) by small cycles has been studied in the literature (see next section).

But here there is another constraint due to the fact that the requests have to be routed on the physical network \( G \), which can be modelled as follows (one can think that to each subnetwork, here cycle, we associate a wavelength): for each subgraph \( I_k \) of the covering there should exist edge disjoint routing in \( G \) i.e. the paths associated in \( G \) to any pair of requests of \( I_k \) should be edge disjoint. We call this property the disjoint routing constraint (DRC). As an illustration, let \( G = C_4 \) and \( I = K_4 \) (See Figure 1). A first covering is given by the two \( C_4 \)'s \((1,2,3,4,1)\) and \((1,3,4,2,1)\) (See Figure 1.(c)), but there does not exist an edge disjoint routing for the cycle \((1,3,4,2,1)\), as it is impossible to associate the requests \((1,3)\) and \((2,4)\) to edge disjoint paths in \( G \). In counterpart, the covering given in Figure 1.(d) by the \( C_4 \) \((1,2,3,4,1)\) and the two \( C_3 \)'s \((1,2,4,1)\) and \((1,3,4,1)\) satisfy the edge disjoint routing property.

Note that if a covering of \( I \) by triangles \( C_3 (= K_3) \) is wanted, the DRC is satisfied as soon as \( G \) is Hamiltonian or 3-connected. However 2-connectivity is not enough as shown by the example of Figure 2.(a) where it is impossible to satisfy DRC for the triangle \( ABC \). Furthermore, a covering by \( C_k \) satisfies DRC if \( G \) is \( k \)-connected; indeed in a \( k \)-connected graph there is always a cycle containing \( k \) given vertices.

When \( G \) is the cycle \( C_n \), where the vertices are labelled with integers modulo \( n \), represented by the set \( \{0, 1, \ldots, n-1\} \), a \( C_k \) satisfies DRC if and only if its vertices can be ordered cyclically modulo \( n \), that is if the vertices can be written \((a_1, a_2, \ldots, a_k)\) with \( 0 \leq a_1 \leq a_2 \leq \cdots \leq a_k \leq n-1 \). As an example, in Figure 2.(b), the cycle \((0, 2, 3, 6, 0)\) satisfies DRC, but the cycle \((0, 4, 3, 6, 0)\) does not satisfy it.

The general problem can be summarized as follows: Find a covering of the edges of a logical graph \( I \) by subgraphs \( I_k \), such that, for each \( I_k \), there exists in the physical graph \( G \) a disjoint routing of the edges of \( I_k \) and such that the cost of
Figure 1: Cycle covering example.
the network is minimized.

The aim is to minimize the cost of the network; that is a very complex function depending of the size of the ADM (Add and Drop Multiplexer) put in each node, the number of wavelengths (associated to the subnetworks) in transit in each optical node and a cost of regeneration and amplification of the signal. When the physical graph is a ring that corresponds to minimize the number of subgraphs \( I_k \) in the covering (as there is a unique physical path associated to a request). Furthermore it reduce the complexity (in terms of number of sub-networks) of the whole network. Moreover, a ring network is not able to support more than a single failure as it is only 2-connected.

Here we solve the problem when the physical graph \( G \) is the cycle of length \( n, C_n \), \( I \) is the complete graph \( K_n \) and the \( I_k \) are cycles \( C_k \) of length \( k \).

3 Results without disjoint routing constraint

Our problem has not yet been studied in the literature; however, without the disjoint routing constraint, it is equivalent to find a covering of the edges of a graph \( I \) by subgraphs \( I_k \). When \( I = K_n \) and \( I_k = K_k \), this problem is known as the covering design problem (See the surveys of Stinson [15] and Mills and Mullin [12]). Moreover, the problem of finding a perfect covering of the edges is related to the existence of an \((n, k, 1)\)-design. In particular, for the covering by 3-cycles, we have the following results (See [13, 12]):

**Theorem 1** [13, 12] The minimum number of 3-cycles required to cover the edges of \( K_n \) is \( \left\lceil \frac{n}{2} \left\lceil \frac{n-1}{3} \right\rceil \right\rceil \).

When \( n \equiv 1 \) or \( 3 \mod 6 \), the edges of \( K_n \) can be partitioned into \( \frac{n(n-1)}{6} \) 3-cycles, also called Steiner triples.
When $I_k$ is isomorphic to $C_k$, the partition problem of the edges of $K_n$ in $C_k$ is known as cycle designs (See [11] for a survey). Also, the covering of the edges of $K_n$ by $C_k$, $k > 3$, was considered in [3], where one can found the following result:

**Theorem 2** [3] The minimum number of 4-cycles required to cover the edges of $K_n$ is

$$\left\lceil \frac{n}{4} \left\lceil \frac{n-1}{2} \right\rceil \right\rceil + \varepsilon(n)$$

with $\varepsilon(n) = \begin{cases} 1 & \text{if } n \equiv 3 \mod 8 \\
0 & \text{otherwise.} \end{cases}$

Finally, the partitioning problem of the edges of $K_n$ by graphs isomorphic to a given graph $H$ is considered in [4, 9].

### 4 Our results

Here, we consider covering of the edges of $K_n$ by cycles satisfying the disjoint routing constraint. Such a covering will be called a DRC-covering of $K_n$.

We will denote by $\rho(n)$ the minimum number of cycles needed in a DRC-covering of $K_n$.

#### 4.1 Lower bounds

**Theorem 3** $\rho(2p+1) \geq \frac{p(p+1)}{2}$ with $p \geq 1$, and $\rho(2p) \geq \frac{p^2+1}{2}$, $p \geq 2$.

**Proof.** Let $C_k'$ be a $k$-cycle of DRC-covering of $K_n$. The disjoint routing property implies that its vertices are cyclically ordered modulo $n$. Thus $C_k'$ can be written $(a_i', a_{i+1}', \ldots, a_k', a_1')$, with $0 \leq a_i' \leq a_{i+1}' \leq \cdots \leq a_k' \leq n-1$.

Let $\delta_i' = a_{i+1}' - a_i'$, $1 \leq i < k-1$, and $\delta_k' = n + a_1' - a_k'$. The disjoint routing property implies $\sum_i \delta_i' = n$.

For an edge $(x, y)$ of $K_n$ with $x < y$, we call difference of the edge the value $y - x$ if $y - x \leq n/2$ or $x + n - y$ otherwise (it corresponds to the distance between $x$ and $y$ on a cycle of length $n$).

In the odd case, $n = 2p+1$, the covering must contain the $n$ edges of difference $d$ for every $d$, $1 \leq d \leq p$. Each difference correspond to an $\delta_i'$, with $\delta_i' = d$ or $n - d$. Thus, $\sum_{i,j} \delta_{ij} \geq \sum_{d=1}^{p} nd = n\frac{p(p+1)}{2}$. Remind that $\sum_i \delta_i' = n$.

Consequently, if the covering contains $\rho(n)$ cycles, we have $np(n) \geq n\frac{p(p+1)}{2}$ and finally, $\rho(n) \geq \frac{p(p+1)}{2}$.

In the even case, $n = 2p$, the covering must contain $n$ edges of difference $d$, $1 \leq d \leq p - 1$ and $\frac{p}{2} = p$ edges of difference $p$. Furthermore, since the degree of the nodes in $K_n$ is odd (equal to $n-1$) and the degree of the nodes of a cycle is even (equal to 2), the covering must contain extra edges (i.e. in each vertex,
there is an edge covered at least twice). Thus, there are at least $\frac{n}{2}$ extra edges of difference at least 1 in the covering (corresponding to a perfect matching). Consequently, $\sum_{i,j} \delta^i_j \geq \left( \sum_{d=1}^{p-1} nd \right) + pp + p = p(p^2 + 1)$ and if the covering contains $p(n)$ cycles, we obtain $np(n) = 2pp(n) \geq p(p^2 + 1)$ and finally $p(n) \geq \frac{p^2 + 1}{2}$. □

Note that for both odd and even cases, the length of the cycles involved in the DRC-covering of $K_n$ has no influence on the lower bound of $p(n)$. Thus, as it will be confirm in the following, small cycles are sufficient to reach the lower bound.

### 4.2 Minimum DRC covering

**Theorem 4** When $n = 2p + 1$, $p(n) = \frac{p(p+1)}{2}$. Furthermore, the DRC-covering of $K_{2p+1}$ consists of $p$ $C_3$ and $\frac{p(p-1)}{2}$ $C_4$.

**Proof.** (By induction on $p$) $K_3$ is covered using one $C_3$. Thus, the theorem is true when $p = 1$.

Suppose now that the theorem is true for $K_{2p+1}$. We will show that it is also true for $n = 2p + 3$. For that, let us arrange the vertices of $K_{2p+3}$ in the following order: $A, 0, 1, \ldots, p - 1, B, p, \ldots, 2p$.

We build a DRC-covering of $K_{2p+3}$ from a DRC-covering of $K_{2p+1}$ as follows. The cycles of the DRC-covering of $K_{2p+3}$ will be

- the $p(p+1)/2$ cycles of a DRC-covering of the $K_{2p+1}$ on vertices $0, 1, \ldots, p - 1, p, \ldots, 2p$,
- the $p$ $C_4$ of a DRC-decomposition of the $K_{2p+1}$ constructed between vertices $0, \ldots, p - 1, p + 1, \ldots, 2p$ on one side and vertices $A$ and $B$ on the other side, namely the $C_4 (A, i, B, p + 1 + i, A)$, $0 \leq i \leq p - 1$,
- the $C_3 (A, B, p, A)$.

One can check that each edge of $K_{2p+3}$ is covered by exactly one of these cycles and altogether we have $p(p + 1)/2 + p + 1 = (p + 1)(p + 2)/2$ cycles. Furthermore, there are exactly $p + 1$ $C_3$ and $p(p + 1)/2$ $C_4$. □

In Figure 3, we show a covering of $K_5$ obtained in that way. Let us called the vertices of $K_5 A, 0, B, 1, 2$ in that order. The DRC-covering of $K_5$ consists of the unique cycle $(0, 1, 2, 0)$ of the covering of $K_3$, plus the $C_4 (A, 0, B, 2, A)$ of a DRC-decomposition of the $K_{2,2}$ constructed between vertices $A$ and $B$ and vertices 0 and 2, plus the $C_3 (A, B, 1, A)$.

**Theorem 5** When $n = 2p$, $p \geq 3$, $p(n) = \left\lceil \frac{p^2 + 1}{2} \right\rceil$. Furthermore, when $n = 4q$, the DRC-covering of $K_{4q}$ consists of 4 $C_3$ and $2q^2 - 3$ $C_4$, and when $n = 4q + 2$ the DRC-covering of $K_{4q+2}$ consists of 2 $C_3$ and $2q^2 + 2q - 1$ $C_4$. 
In order to prove this theorem, we first need to prove some lemmas.

**Lemma 1** $K_6$ can be covered by 2 $C_3$ and 3 $C_4$.

**Proof.** The covering is given by the two $C_3$: (0, 1, 3, 0) and (0, 1, 4, 0), plus three $C_4$: (0, 2, 4, 5, 0), (1, 2, 3, 5, 1) and (2, 3, 4, 5, 2), as shown in Figure 4. Furthermore, there are three edges, (0, 1), (2, 3) and (4, 5), covered exactly twice (they form a perfect matching). \qed

![Figure 4: $K_6$](image)

**Lemma 2** If there exists a DRC-covering of $K_{4q+2}$ with $\rho(4q + 2) = 2q^2 + 2q + 1$ cycles, then there exists a DRC-covering of $K_{4q+4}$ with $\rho(4q + 4) = 2q^2 + 4q + 3$ cycles.

**Proof.** Let us call and range the vertices of $K_{4q+4}$ in the following order: $A, 0, 1, \ldots, 2q, B, 2q + 1, \ldots, 4q + 1$.

We build a DRC-covering of $K_{4q+4}$ from a DRC-covering of $K_{4q+2}$ as follows. The cycles of the DRC-covering of $K_{4q+4}$ will be
• the $2q^2 + 2q + 1$ cycles of a DRC-covering of the $K_{4q+2}$ on vertices $0, 1, \ldots, 2q, 2q + 1, \ldots, 4q + 1$,

• the $2q$ $C_4$ of a DRC-decomposition of the $K_{4q, 2}$ constructed on vertices $1, \ldots, 2q, 2q + 1, \ldots, 4q$ on one side, and vertices $A$ and $B$ on the other side, namely, the $C_4 (A, i, B, 2q + i, A), 1 \leq i \leq 2q$.

• the $2$ triangles $(A, 0, B, A)$ and $(A, B, 4q + 1, A)$.

One can check that every edge of $K_{4q+4}$ is covered by one of these cycles and that altogether we have $2q^2 + 2q + 1 + 2q + 2 = 2q^2 + 4q + 3 = \left[ \frac{(2q+2)^2+1}{2} \right]$ cycles. Furthermore, there are exactly $4$ $C_3$ in the covering ($2$ from the DRC-covering of $K_{4q+2}$ and the $2$ extra $C_3$).

To illustrate this proof, we indicate in Figure 5 the cycles involved in the covering of $K_8$.

- **Figure 5:** Cycles involved in the covering of $K_8$.

**Lemma 3** If there exists a DRC-covering of $K_{4q+2}$ with $\rho(4q+2) = 2q^2 + 2q + 1$ cycles, then there exists a DRC-covering of $K_{4q+6}$ with $\rho(4q+6) = 2q^2 + 6q + 5$ cycles.

**Proof.**

We will prove a little stronger theorem, imposing some extra properties in the decomposition which will be kept in the construction.

Let us suppose that there exists a DRC-covering of $K_{4q+2}$, where the nodes $0, 1, \ldots, 4q + 1$ are cyclically ordered, with the following properties:

• the edges (of the perfect matching) $(0, 1), (2, 3), \ldots, (4q, 4q + 1)$ are covered exactly twice, while other edges are covered exactly once.
Let us call and range the vertices of $K$ either 3 or 4.

Note that these properties are satisfied by the covering of $K_6$ of Lemma 1 (x being either 3 or 4).

Let us call and range the vertices of $K_{4q+6}$ in the following order:

$0, A, B, 1, \ldots, 2q+1, C, D, 2q+2, \ldots, 4q+1$. The cycles of the DRC-covering of $K_{4q+6}$ will be

- the $2q^2 + 2q$ cycles of the covering of $K_{4q+2}$ except the 3-cycle $(0, 1, x, 0)$,
- the $2q$ 4-cycles $(A, i, C, f(i), A)$, with $2 \leq i \leq 2q + 1$ and where $f$ is a bijection from $\{2, 3, \ldots, 2q + 1\}$ to $\{2q + 2, \ldots, 4q + 1\}$,
- the $2q + 1$ 4-cycles $(B, j, D, g(j), B)$, $1 \leq j \leq 2q + 1$, and where $g$ is a bijection from $\{1, 2, \ldots, 2q + 1\}$ to $\{2q + 2, \ldots, 4q + 1, 0\}$,
- the 3 $C_4$ $(A, B, C, D, A)$, $(0, A, 1, x, 0)$, $(B, 1, C, D, B)$ and the $C_3$ $(0, A, C, 0)$.

One can check that each edge of $K_{4q+6}$ is covered by one of these cycles and that altogether, we have $2q^2 + 2q + 2q + 2q + 1 + 3 + 1 = 2q^2 + 6q + 5 = \left\lceil \frac{(2q + 2)^2 + 1}{2} \right\rceil$ cycles. Furthermore, there are still exactly 2 $C_3$ in the covering. Also, the edges $(0, A), (B, 1), (2, 3), \ldots, (2q, 2q + 1), \ldots, (C, D), (2q + 2, 2q + 3), \ldots, (4q, 4q + 1)$ (corresponding to a perfect matching) are covered twice, while other edges are covered only once. Moreover, the edge $(0, A)$, which is covered twice, appears in the 3-cycle $(0, A, C, 0)$.

Now, we are able to prove Theorem 5.

Proof. of Theorem 5 (By induction)

The theorem is true for $n = 6$ as shown by Lemma 1. Note that the covering of $K_6$ satisfies the two extra properties needed in the proof of Lemma 3. So using Lemma 3, one can build by induction the DRC-covering of $K_{4q+2}$, $q \geq 1$, by $p(4q + 2) = 2q^2 + 2q + 1$ cycles. Then, using Lemma 2, one can build the DRC-covering of $K_{4q+4}$, $q \geq 1$, by $p(4q + 4) = 2q^2 + 4q + 3$ cycles.

So Theorem 5 is proved.

5 Conclusion

The problem of the design of a survivable WDM network was considered as an extension of the classical edge covering problem by addition of the disjoint routing constraint. In particular, we have studied the case of a physical ring network with the all-to-all ($K_n$) communication instance. For this design problem, we give a solution with the optimal number of sub-network (cycles). As an extension of
this problem, we are now investigating cases with other communication instances such as $\lambda K_n$ (or more general logical graphs). We also think to consider other network topologies for example trees of rings, grids or tori. Moreover it will be interesting to consider in these extensions the real cost function.

References


Jean-Claude Bermond is with the MASCOTTE project, CNRS-I3S-INRIA, 2004, route des Lucioles, B.P. 93, F-06902 Sophia-Antipolis Cedex.
E-mail: jean-claude.bermond@sophia.inria.fr

Lilian Chacon is with France Telecom R&D, Technopole Anticipa, 2, avenue Pierre Marzin, F-22307 Lannion Cedex.
E-mail: lilian.chacon@rd.francetelecom.fr

David Coudert is with the MASCOTTE project, CNRS-I3S-INRIA, 2004, route des Lucioles, B.P. 93, F-06902 Sophia-Antipolis Cedex.
E-mail: david.coudert@sophia.inria.fr

François Tillerot is with France Telecom R&D, Technopole Anticipa, 2, avenue Pierre Marzin, F-22307 Lannion Cedex.
E-mail: francois.tillerot@rd.francetelecom.fr
Competitive Online Routing in Geometric Graphs*

PROSENJIT BOSE
Carleton University, Canada

PAT MORIN
McGill University, Canada

Abstract

We consider online routing algorithms for finding paths between the vertices of plane graphs. Although it has been shown in Bose et al. [3] that there exists no competitive routing scheme that works on all triangulations, we show that there exists a simple online $O(1)$-memory $c$-competitive routing strategy that approximates the shortest path in triangulations possessing the diamond property, i.e., the total distance travelled by the algorithm to route a message between two vertices is at most a constant $c$ times the shortest path. Our results imply a competitive routing strategy for certain classical triangulations such as the Delaunay, greedy, or minimum-weight triangulation, since they all possess the diamond property.

Keywords
online routing, competitive routing, geometric graph, minimum weight triangulation, delaunay triangulation, greedy triangulation

1 Introduction

Path finding, or routing, is central to a number of fields including geographic information systems, urban planning, robotics, and communication networks. In many cases, knowledge about the environment in which routing takes place is not available beforehand, and the vehicle/robot/packet must learn this information through exploration. Algorithms for routing in these types of environments are referred to as online [2] routing algorithms.

In this paper we consider online routing in the following abstract setting [4]: The environment is a plane graph, $G$ (i.e., the planar embedding of $G$) with $n$ vertices and whose edges are weighted with the Euclidean distance between their

---

*This research was partly funded by the Natural Sciences and Engineering Research Council of Canada.
endpoints. The source $s$ and destination $t$ are vertices of $G$, and a packet can only travel on edges of $G$. Initially, a packet only knows the coordinates of $s$, $t$, and $N(s)$, where $N(v)$ denotes the set of vertices adjacent to a node $v$. When a packet visits a node $v$, it learns the coordinates of $N(v)$.

Bose and Morin [4] classify routing algorithms based on their use of memory. A deterministic routing algorithm is memoryless or oblivious if, given a packet currently at vertex $v$ and destined for node $t$, the algorithm decides where to forward the packet based only on the coordinates of $v$, $t$ and $N(v)$. An $O(1)$-memory routing algorithm decides where to move a packet based only on the coordinates of $v$, $t$, $N(v)$, and the content of its constant size memory $M$.

We say that a routing algorithm $A$ is defeated by a graph $G$ if there exists a pair of vertices $s, t \in G$ such that a packet stored at $s$ will never reach $t$ when being routed using $A$. Otherwise, we say that $A$ works for $G$.

Let $A(G, s, t)$ denote the length of the walk taken by routing algorithm $A$ when travelling from vertex $s$ to vertex $t$ of $G$, and let $SP(G, s, t)$ denote the length of the shortest path, in $G$, between $s$ and $t$. We say that $A$ is $c$-competitive for a class of graphs $G$ if

$$\frac{A(G, s, t)}{SP(G, s, t)} \leq c$$

for all graphs $G \in G$ and all $s, t \in G$, $s \neq t$. We say that $A$ is simply competitive if $A$ is $c$-competitive for some constant $c$.

Recently, several papers have dealt with online routing and related problems in geometric settings. Kalyanasundaram and Pruhs [7] give a 16-competitive algorithm to explore any unknown plane graph, i.e., visit all of its nodes. This online exploration problem makes the same assumptions as those made here, but the goal of the problem is to visit all vertices of $G$, not just $t$. This difference leads to inherently different solutions.

Kranakis et al. [8] give a deterministic oblivious routing algorithm that works for any Delaunay triangulation, and give a deterministic $O(1)$ memory algorithm that works for any connected plane graph.

Bose and Morin [4] also study online routing in geometric settings, particularly triangulations. They give a randomized oblivious routing algorithm that works for any triangulation, and ask whether there is a deterministic oblivious routing algorithm for all triangulations. They also give a competitive $O(1)$-memory routing algorithm for Delaunay triangulations.

Cucka et al. [5] experimentally evaluate the performance of routing algorithms very similar to those described by Kranakis et al. [8] and Bose and Morin [4]. When considering the Euclidean distance travelled during point-to-point routing, their results show that the GREEDY routing algorithm [4] performs better than the

\[ \text{A constant size memory can hold a constant number of vertex identifiers, distances, and } O(\log n) \text{ bit integers.} \]
COMPASS routing algorithm [4, 8] on random graphs, but does not do as well on Delaunay triangulations of random point sets. However, when one considers not the Euclidean distance, but the number of edges traversed (link distance), then the COMPASS routing algorithm is slightly more efficient for both random graphs and Delaunay triangulations. Recently, Bose et al. [3] provide a deterministic oblivious routing strategy that works for all triangulations. However, they also show that there is no competitive online routing algorithm under the Euclidean distance metric in arbitrary triangulations. In light of this fact, it is interesting to classify which types of triangulations admit competitive routing algorithms since it was shown in [4] that there exist $O(1)$-memory competitive routing strategies for the Delaunay triangulation.

In this paper we explore this question further and present an $O(1)$-memory competitive routing strategy that works for the class of triangulations possessing the diamond property. This class is fairly large as it includes such classical triangulations as the Delaunay, greedy and minimum-weight triangulations.

The remainder of the paper is organized as follows: In Section 2 we present a deterministic competitive online routing algorithm for routing on triangulated polygons with two ears. Section 3 presents our results for routing on triangulations that possess the diamond property. Finally, Section 4 summarizes and concludes with open problems.

2 Competitive Routing in Triangulated Polygons with Two Ears

Before addressing the problem of routing on plane graphs, we first study the problem in a specific setting that will prove to be quite useful in the sequel. A triangulated simple polygon is a geometric outer-planar graph $P$ where every face except the outer face is a triangle. A vertex of degree two in $P$ is known as an ear. In this section, we study triangulated simple polygons with only two ears, $s$ and $t$. Given such a graph $P$, we devise a simple online $O(1)$-memory routing strategy that finds a path from $s$ to $t$ such that the total distance travelled by the algorithm when routing from $s$ to $t$ is at most $9 \cdot SP(P, s, t)$ (i.e., the shortest path from $s$ to $t$ in $P$).

The two ears naturally divide the outer face of $P$ into two chains (see Figure 1). Let $\{s = a_0, a_1, \ldots, a_m = t\}$ be the sequence of vertices in the upper chain and $\{s = b_0, b_1, \ldots, b_n = t\}$ be the sequence of vertices in the lower chain. If the shortest path from $s$ to $t$ happened to be one of these two chains, then one could devise a simple online routing strategy by directly apply a result of Baeza-Yates et al. [1]. Baeza-Yates et al. studied the following problem: given a two-way infinite line and a searcher starting at the origin, the searcher must find a goal that lies

---

2Cucka et al. call these algorithms P-DFS and D-DFS, respectively.
at some unknown distance $d$ from the origin. The searcher can only move in unit steps and the objective is to minimize the ratio of the distance traversed to the true distance $d$. The strategy proposed in [1] is to have the searcher alternate her search between the two sides of the origin and each time the searcher travels a certain distance on one side of the origin, she doubles the distance travelled on the other side of the origin. This results in the searcher travelling at most $9d$ steps to find the goal. By having the upper and lower chain represent each of the two sides of the origin, applying this technique would result in a 9-competitive search strategy. Unfortunately, the shortest path need not be one of the two chains. In fact, the ratio between the length of the shortest path and either of the two chains can be unbounded (see Figure 2).

We circumvent this problem by uncovering some key properties of the shortest-path tree of $P$ rooted at $s$, denoted $T(s)$. The tree $T(s)$ is the tree formed by taking the union of the shortest paths from $s$ to all the vertices in $P$. The shortest path from $s$ to a node $x$ in $P$ consists of a sequence of nodes from the upper and lower chain. This sequence cannot have a node from the lower chain between two consecutive nodes in the upper chain or vice versa.

We refer to nodes of degree 1 in $T(s)$ as leaves, nodes of degree 2 as internal nodes and all other nodes as branching nodes. The crucial observation is that the shortest path from $s$ to $t$ visits every branching node.

**Lemma 1** Given a triangulated simple polygon $P$ with two ears $s$ and $t$, the short-
Figure 2: The paths along the lower and upper chains of $P$ can be arbitrarily long.

**Proof.** The proof is by induction on the number of vertices in $P$. The lemma holds trivally when $P$ has 4 vertices. Assume that the lemma holds when $P$ has $4 \leq k < n$ vertices. Vertex $t$ has degree 2 and is adjacent to one vertex, $a_i$, from the upper chain and one vertex $b_j$ from the lower chain. Remove $t$ from $P$ to form a triangulated polygon $P'$. Now, $P'$ has only two ears, $s$ and one of $a_i$ or $b_j$. Without loss of generality, assume $b_j$ is an ear of $P'$. By the inductive hypothesis, the shortest path from $s$ to $b_j$ visits every branching node of the shortest path tree rooted at $s$. By re-introducing $t$, the shortest path tree does not change very much. In fact, $t$ is a leaf in the tree and it is adjacent either to $a_i$ or $b_j$. However, the least common ancestor of $a_i$ and $b_j$ in the tree is the first branching node in path from $b_j$ to $s$. Therefore, $t$ visits every branching node of $T(s)$. □

Branching nodes can be identified locally with only a constant amount of extra information. Consider the node $a_i$ in the upper chain. Let $b_j, b_{j+1}, \ldots, b_k$ be the sequence of nodes in the lower chain adjacent to $a_i$. If we know the length of $SP(P, s, a_{i-1})$ and $SP(P, s, b_j)$, then we can identify whether $a_i$ or any of its adjacent vertices on the lower chain are branching nodes. For example, the node $b_j$ is a branching node if $|SP(P, s, b_j)| + \text{dist}(b_j, a_i) < |SP(P, s, a_{i-1})| + \text{dist}(a_{i-1}, a_i)$, where $\text{dist}(p, q)$ represents the Euclidean distance between points $p$ and $q$.

The approach to finding a competitive routing algorithm is to move from branching node to branching node in a competitive fashion. Notice that to find a short path between two consecutive branching nodes, we only need to explore two paths, one consisting solely of upper chain vertices and the other of lower chain vertices. The following algorithm, which we call NEXT-BRANCH starts at a branching node $x$ and moves to the next branching node $y$ travelling a total of $9 \cdot SP(P, x, y)$. Since $x$ is a branching node, there are two paths of $T(s)$ leading out of $x$. One of them leads to $y$ and the other ends at a leaf. Without loss of generality, let $P_1 = x, a_i, a_{i+1}, \ldots, a_j, y$ be one of the paths and $P_2 = x, b_k, b_{k+1}, \ldots, b_l$ be the other.
Next-Branch

1: \[ d = \min \{ \text{dist}(x, a_i), \text{dist}(x, b_k) \} \]

2: Repeat

3: travel along \( P_1 \) until reaching a branching node or until reaching a vertex \( a_z \) such that the length of the path from \( x \) to \( a_{z+1} > d \). When travelling from one vertex to the next, retain the length of the shortest path to the one upper chain and one lower chain vertex required to compute all shortest path information at the next step. If a branching node is reached quit, otherwise return to \( x \).

4: \[ d \leftarrow 2d \]

5: travel along \( P_2 \) until reaching a branching node or until reaching a vertex \( b_z \) such that the length of the path from \( x \) to \( b_{z+1} > d \). When travelling from one vertex to the next, retain the length of the shortest path to the one upper chain and one lower chain vertex required to compute all shortest path information at the next step. If a branching node is reached quit, otherwise return to \( x \).

6: \[ d \leftarrow 2d \]

7: Until \( y \) is reached

We note that in steps 3 and 5 of the algorithm, it is not necessary to return to \( x \) upon an unsuccessful search. In fact, it would be easier to simply follow an edge connecting a vertex of the upper (resp. lower) chain to a vertex on the lower (resp. upper) chain. Although this shortcuitting will likely reduce the distance travelled in practice, it is difficult to reduce the upper bound when these short cuts are taken and the analysis is much simpler if one returns to \( x \) after each unsuccessful search.

Lemma 2 Starting at \( x \), Next-Branch reaches \( y \) after travelling a total of \( 8 \cdot SP(P,x,y) \).

Proof. Let \( c = \min\{ \text{dist}(x, a_i), \text{dist}(x, b_k) \} \). Let \( d_f = 2^k c \) be the value of \( d \) during the final exploration step (Line 3 or Line 5) of the algorithm. Therefore, the total distance travelled by the algorithm is equal to

\[
D = 2 \cdot \sum_{i=1}^{k-1} 2^i c + L \\
\leq 2^{k+1} c + L
\]

where \( L \) is the distance travelled during the last exploration step. There are now two cases to consider.

Case 1: The algorithm terminated while exploring the shorter of the two paths \( P_1 \) or \( P_2 \). Then \( d_f \leq 4 \cdot \min\{ \text{length}(P_1), \text{length}(P_2) \} \), otherwise the algorithm would have reached \( y \) in the previous iteration of the algorithm. Therefore

\[
D \leq 8 \cdot \min\{ \text{length}(P_1), \text{length}(P_2) \} + L \\
= 9 \cdot \min\{ \text{length}(P_1), \text{length}(P_2) \}
\]
Case 2: The algorithm terminated while exploring the longer of the two paths $P_1$ or $P_2$. Then $x \leq d_f \leq 2 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\}$, otherwise the algorithm would have reached $y$ in the previous exploration step. Then

$$D \leq 4 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} + L$$

$$\leq 6 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\}$$

In both cases, the conditions of the lemma are satisfied. □

Putting Lemma 1 and Lemma 2 together, we devise FIND-SHORT-PATH, an online competitive $O(1)$ memory routing strategy to move from $s$ to $t$ in $P$. Starting at vertex $s$, repeatedly invoke NEXT-BRANCH until $t$ is reached.

**Theorem 1** FIND-SHORT-PATH reaches $t$ after having travelled at most $9$ times $SP(P, s, t)$.

**Proof.** By Lemma 1, the shortest path from $s$ to $t$ must visit every branching node. Since each of these steps is 9-competitive, by Lemma 2, the theorem follows. □

### 3 Competitive Routing in Triangulations

Although it was shown in [3] that there is no competitive online routing algorithm under the Euclidean distance metric in arbitrary triangulations, in this section we provide an $O(1)$-memory competitive algorithm for the class of triangulations possessing the diamond property. Das and Joseph [6] showed these triangulations approximate the complete Euclidean graph in terms of the shortest path length. We elaborate on the precise definition of this property.

Let $\alpha$ be any angle $0 < \alpha \leq \pi/2$. For an edge $e$ of a triangulation $T = (V, E)$, consider the two isosceles triangles $t_1$ and $t_2$ whose base is $e$ and with base angle $\alpha$. Refer to Fig. 3. The edge $e$ satisfies the diamond property with parameter $\alpha$ if one of $t_1$ or $t_2$ does not contain any point of $V$ in its interior. A triangulation $T$ satisfies the diamond property with parameter $\alpha$ if every edge of $T$ satisfies the diamond property with parameter $\alpha$. Das and Joseph prove the following.

**Lemma 3** [6] Given a triangulation $T = (V, E)$ satisfying the diamond property with parameter $\alpha$, there exists a constant $d_\alpha$ (depending on $\alpha$), such that $\forall x, y \in V, SP(T, x, y)/\text{dist}(x, y) \leq d_\alpha$.

They showed that the diamond property is not an obscure property that is possessed by only a few triangulations but that the class of triangulations possessing the diamond property is fairly rich and includes some of the classical triangulations.
Figure 3: The edge $e$ satisfies the diamond property if at least one of $t_1$ and $t_2$ does not contain any point of $V$ in its interior.

Figure 4: The graph $T_{xy}$ (shaded).

**Lemma 4** [6] The set of triangulations satisfying the diamond property include such classical triangulations as the Delaunay triangulation, the minimum weight triangulation and the greedy triangulation.

Given two vertices $x, y$ in a triangulation $T$, consider the set $S_{xy}$ of triangles of $T$ whose interiors intersect the line segment $[x, y]$. Define $T_{xy}$ as the subgraph of $T$ containing only those edges of $T$ bounding triangles of $S_{xy}$. By construction, $T_{xy}$ is a triangulated simple polygon with two ears, $x$ and $y$.\(^3\) An example is shown in Fig. 4.

**Lemma 5** Given a triangulation $T = (V, E)$ satisfying the diamond property with parameter $\alpha$, and two vertices $x, y \in V$, the shortest path between $x$ and $y$ in $T_{xy}$ is at most $d_\alpha$ times $\text{dist}(x, y)$.

**Proof.** Essentially, although it is not stated explicitly, a careful analysis of the proof of Lemma 3 reveals that in fact, for every triangulation satisfying the diamond property with parameter $\alpha$, the shortest path between $x$ and $y$ in $T_{xy}$ has

\(^3\)There are degenerate cases in which vertices other than $x$ and $y$ lie on the segment $[x, y]$. In such cases, $T_{xy}$ is a sequence of triangulated simple polygons joined at the vertices on $[x, y]$. 
length at most \( d_\alpha \) times the line segment \([x, y]\). This is stronger than simply the shortest path, but the distance is related to the Euclidean distance.

Note that the above lemma shows that the path is related not only to the shortest path between \( x \) and \( y \) but in fact to the Euclidean distance between \( x \) and \( y \). In order to route in a competitive fashion between vertices \( x, y \) in a triangulation \( T \) possessing the diamond property, attention can be restricted to the subgraph \( T_{xy} \). Given a vertex \( z \) of \( T \), a local test determines whether or not \( z \) lies in \( T_{xy} \) by testing whether any of the edges adjacent to \( z \) intersect the segment \([x, y]\). This leads to a routing scheme based on algorithm \textproc{FIND-SHORT-PATH}. Basically, starting at vertex \( x \), run \textproc{FIND-SHORT-PATH} until \( y \) is reached. At each step in the algorithm, a local test allows one to determine which vertices form the upper and lower chain.

We conclude with the following:

**Theorem 2** Given a triangulation \( T = (V, E) \) satisfying the diamond property with parameter \( \alpha \), and two vertices \( x, y \in V \), a modified \textproc{FIND-SHORT-PATH} is an \( O(1) \)-memory online competitive routing algorithm that moves a packet from \( x \) to \( y \) after travelling a total of at most \( 9 \cdot d_\alpha \cdot \text{dist}(x, y) \).

### 4 Conclusions

Given that no competitive routing strategy works for all triangulations, in this paper we presented an \( O(1) \)-memory competitive routing strategy that works for the class of triangulations possessing the diamond property. This class is fairly large as it includes such classical triangulations as the Delaunay, greedy and minimum-weight triangulations. The routing strategy is based on a simple online competitive strategy for routing on triangulated simple polygons.

These results are in contrast with results for the link distance metric, where the length of a path is the number of edges it uses. It is known [3] that no competitive algorithm exists for greedy, minimum-weight, or Delaunay triangulations under this metric. This raises the question: For what classes of geometric graphs do competitive routing algorithms exist under the link distance metric?

### References


Prosenjit Bose is with the School of Computing Science, Carleton University, Ottawa, ON, Canada K1S 5B6. E-mail: jit@scs.carleton.ca

Pat Morin is with the School of Computer Science, McGill University, Montreal, PQ, Canada. E-mail: morin@cs.mcgill.ca
Parallel Real-Time Complexity:
A Strong Infinite Hierarchy*

STEFAN D. BRUDA
Queen's University, Canada

SELM G. AKL
Queen's University, Canada

Abstract
We present a new complexity theoretic approach to real-time parallel computations. Based on the theory of timed $\omega$-languages, we define complexity classes that capture the intuitive notion of resource requirements for real-time computations in a parallel environment. Then, we show that, for any positive integer $n$, there exists at least one timed $\omega$-language $L_n$ which is accepted by a $2n$-processor real-time algorithm using arbitrarily slow processors, but cannot be accepted by a $(2n-1)$-processor real-time algorithm. It follows therefore that real-time algorithms form an infinite hierarchy with respect to the number of processors used. This result is strong, in the sense that it holds for any model of parallel computation. Our result motivates (and initiates) the development of a parallel real-time complexity theory.

Keywords
parallel computing, real-time, complexity theory, timed $\omega$-languages

1 Introduction

Question 1 Can one find any problem that is solvable by an algorithm that uses $n$ processors, $n > 1$, and is not solvable by a sequential algorithm, even if this sequential algorithm runs on a machine whose (only) processor is $n$ times faster than each of the $n$ processors used by the parallel implementation?

Although it is standard to assume that each processor on a parallel computer is as fast as the single processor on the sequential computer used for comparison, Question 1 does make sense in practice. Besides, questions of this kind are crucial for the process of developing a parallel real-time complexity theory. Indeed, a

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.
meaningful such theory should be invariant to secondary issues like the speed of some particular machine. Thus, an answer to the above question is also important from a complexity theoretic point of view.

There are noteworthy results in the area of real-time parallel computation [3, 4, 6, 14], but all of them make the assumption of equal computational speed in the parallel and sequential cases, and thus none of them appear to properly address Question 1. The closest to an answer for this question is probably a result concerning real-time Turing machines, conforming to which a \( k \)-tape real-time Turing machine is strictly more powerful that a \((k-1)\)-tape one [1, 8]. However, the model of choice here creates another problem. Indeed, for one thing, Turing machines appear not to be an expressive enough model for either sequential or parallel real-time computations. On the other hand, this time from a parallel point of view, it is not clear whether the concept of tape in a Turing machine can be equated with the concept of processor in a parallel algorithm. A truly significant result should thus be based on a more realistic model of real-time parallel computation.

As for the choice of such a model, there are some formal models for real-time computations, but many of them fail to take into consideration aspects that are important in practice. The already mentioned real-time Turing machine [18] (the oldest and the most widely studied such a model) cannot express specific deadlines, such as “this computation should take 4 seconds.” The real-time producer/consumer paradigm [13] can express neither real-time events that occur acyclically, nor variable arrival rates for the input. By contrast, the \( \omega \)-regular languages [5] seem to be particularly well suited for modeling real-time problems. Nonetheless, the device used for the recognition of such languages is not sufficiently powerful to take into account all the real-time applications [9]. Finally, timed \( \omega \)-languages appear to be expressive enough in order to capture all the practically important aspects of real-time computations. This model was proposed in [9], together with the thesis conforming to which well-behaved timed \( \omega \)-languages model exactly all real-time computations. Indeed, such a thesis is supported in [9], where it is shown how those ingredients that, when present, give to some problem the “real-time” qualifier (namely, computing with deadlines, and input data that arrive in real-time during the computation) can be modeled using timed \( \omega \)-languages. In addition, such languages are shown to be capable of modeling with reasonable ease practical real-time applications (at least) in the areas of real-time database systems and ad hoc networks [10].

---

**AN INTUITIVE ASIDE.**

On the intuitive level, a positive answer to Question 1 for \( n = 2 \) is provided by (a slightly modified version of) the *pursuit and evasion on a ring* example presented in [2]: An entity \( A \) is in pursuit of another entity \( B \) on the circumference of a circle, such that \( A \) and \( B \) move at the same top speed.
Clearly, A never catches B. Now, if two entities C and D are in pursuit of entity B on the circumference of a circle, such that each of C and D moves at $1/x$ the speed of A (and B), $x > 1$, then C and D always catch B. This modified version of the pursuit/evasion problem was mentioned for the first time in [8].

Starting from this intuition, we lay in this paper the basis for a parallel real-time complexity theory: We start by defining the underlying notions for such a theory, in particular real-time complexity classes, and a notion of input size suitable for this domain. Then, we construct a timed $\omega$-language that models the geometric problem presented above. We extend this language to an “$n$-dimensional circle,” $n \geq 1$, and we show that such a language is accepted by a $2n$-processor PRAM, but there does not exist any $(2n-1)$-processor algorithm that accepts the language. Thus, we prove that the hierarchy of parallel machines solving real-time problems is infinite. To our knowledge, this is the first time such a result is obtained. Finally, we show that the infiniteness of the parallel real-time hierarchy is invariant with the model of parallel computation involved.

True, this is a purely theoretical result. With this respect, one should also notice that the geometrical problem offers an intuition only for the way the timed $\omega$-words are constructed. Although we could not come up with any computational problem that is equivalent with the geometrical one, we believe that such a problem exists, according to the thesis stated in [9]. Still, whether such a problem is of any practical importance, this remains to be seen. Nevertheless, we believe that such a result offers a good motivation for the pursuit of a systematic, complexity theoretical investigation of real-time computations.

2 Preliminaries

Consider some alphabet $\Sigma$. $\Sigma^\omega$ is the set of all finite words $\Sigma$; $\omega$ is the cardinality of $\mathbb{N}$; $\lambda$ is the empty word. $\Sigma^\omega$ contains those words over $\Sigma$ of length $\omega$. Given $A \subseteq \Sigma (a \in \Sigma)$, $|\sigma|_A$ denotes the length of $\sigma$ restricted to $A (\{a\})$. $\Sigma^0 = \{\lambda\}$, and $\Sigma' = \Sigma \times \Sigma^{i-1}$, $i \geq 1$. Given two (infinite or finite) words $\sigma = \sigma_1 \sigma_2 \ldots$ and $\sigma' = \sigma'_1 \sigma'_2 \ldots$, we say that $\sigma'$ is a subsequence of $\sigma$ iff (a) for each $\sigma'_j$ there exists a $\sigma_j$ s.t. $\sigma'_j = \sigma_j$, and (b) for any $i, j, k, l$ s.t. $\sigma'_i = \sigma_j$ and $\sigma'_k = \sigma_l$, it holds that $i > k$ iff $j > l$.

The following summary conforms to [9], with some terminological modifications. A sequence $\tau = \tau_1 \tau_2 \ldots \in \mathbb{N}^\omega$ is a time sequence if it is an infinite sequence of positive values, such that the monotonicity constraint is satisfied: $\tau_i \leq \tau_{i+1}$ for all $i > 0$. A (finite or infinite) subsequence of a time sequence is a time sequence. A well-behaved time sequence is a time sequence $\tau = \tau_1 \tau_2 \ldots$ for which the progress condition also holds: for every $t \in \mathbb{N}$, there exists some finite $i \geq 1$ such that $\tau_i > t$. A time sequence may be finite or infinite; a well-behaved time
sequence is always infinite.

A timed $\omega$-word over $\Sigma$ is a pair $(\sigma, \tau)$, where $\tau \in \mathbb{N}^k$ is a time sequence, $k \in \mathbb{N} \cup \{0\}$, and $\sigma \in \Sigma^k$. Some $\tau_i$ from $\tau$ represents the time at which $\sigma_i$ becomes available as input. A well-behaved timed $\omega$-word is a timed $\omega$-word $(\sigma, \tau)$, where $\tau$ is a well-behaved time sequence. A (well-behaved) timed $\omega$-language over some alphabet $\Sigma$ is a set of (well-behaved) timed $\omega$-words over $\Sigma$.

A real-time algorithm $A$ consists in a finite control (program), an input tape (input stream) that always contains a (not necessarily well-formed) timed $\omega$-word, and an output tape (output stream) containing symbols from some alphabet $\Delta$. The input tape has the same semantics as a timed $\omega$-word: if $(\sigma_i, \tau_i)$ is an element of the input tape, then $\sigma_i$ is available for $A$ at precisely the time $\tau_i$. During any time unit, $A$ may add at most one symbol to the output tape. The content of the output tape of $A$ working on $w$ is denoted by $o(A, w)$. $A$ may have access to an infinite amount of storage space (working tape(s), RAM memory, etc.) outside the input and output tapes, but only a finite amount of this space can be used for any computation performed by the algorithm.

There exists a designated symbol $f \in \Delta$ with the same meaning as the final state from [5]: A real-time algorithm $A$ accepts the timed $\omega$-language $L$ if, on any input $w$, $|o(A, w)|_f = \omega$ iff $w \in L$.

The union, intersection, and complement for timed $\omega$-languages are straightforwardly defined. One can rely on the semantics of timed words in defining a meaningful concatenation operation. Thus, the concatenation of two timed words is defined as the union of their sequences of symbols, ordered in nondecreasing order of their arrival time: Let $(\sigma', \tau')$ and $(\sigma'', \tau'')$ be two timed $\omega$-words over $\Sigma$. Then, $(\sigma, \tau)$ is the concatenation of $(\sigma', \tau')$ and $(\sigma'', \tau'')$ $((\sigma, \tau) = (\sigma', \tau')(\sigma'', \tau''))$, iff (a) $\tau$ is a time sequence; $(\sigma'_1, \tau'_1)(\sigma'_2, \tau'_2) \ldots$ and $(\sigma''_1, \tau''_1)(\sigma''_2, \tau''_2) \ldots$ are sub-sequences of $(\sigma_1, \tau_1)(\sigma_2, \tau_2) \ldots$; for any $i > 0$, there exists $j > 0$ and $d \in \{1, n\}$ such that $(\sigma_i, \tau_i) = (\sigma'_j, \tau'_j)$; (b) for any $d \in \{1, n\}$ and any positive $i$ and $j$, $i < j$, such that $\tau'_i = \tau'_j$ for any $k, l, i \leq k < l \leq j$, there exists $m$ such that, for any $0 \leq t \leq j - i$, $(\sigma_{m+1}, \tau_{m+1}) = (\sigma_{m+i}, \tau_{m+i})$; and (c) for any positive integers $i$ and $j$ such that $\tau'_i = \tau'_j$, there exist integers $k$ and $l$, $k < l$, such that $(\sigma_k, \tau_k) = (\sigma'_i, \tau'_i)$ and $(\sigma_l, \tau_l) = (\sigma''_i, \tau''_i)$.

The two supplementary constraints imposed on the operation of concatenation order the result in the absence of any ordering based on the arrival time, such that nondeterminism is eliminated. One should also note that the concatenation of timed $\omega$-words is associative. Given two timed $\omega$-languages $L_1$ and $L_2$, the concatenation of $L_1$ and $L_2$ is $L_1L_2 = \{w_1w_2|w_1 \in L_1, w_2 \in L_2\}$. The notation $\prod_{i=1}^n w_i (\prod_{i=1}^n L_i)$ is a shorthand for $w_1w_2\ldots w_n (L_1L_2\ldots L_n)$.

Given some timed $\omega$-language $L$, $L^0 = \emptyset$, $L^1 = L$, and $L^k = LL^{k-1}$, $k > 1$. The Kleene closure of $L$ is $L^* = \bigcup_{0 \leq k < \omega} L^k$. For some timed $\omega$-word $w = (\sigma, \tau)$,
detime(w) = σ and time(w) = τ.

**Theorem 1**  The set of (well-behaved) timed ω-languages is closed under intersection, union, complement, concatenation, and Kleene closure, under a proper definition of the latter two operations. A subset of a (well-behaved) timed ω-language is a (well-behaved) timed ω-language.

### 3 Sizing Up Real-Time Computations

**What to measure** Classical complexity theory measures the amount of resources required for the successful completion of some algorithm. Such resources are running time, storage space, and, to a lesser degree, the number of processors used by a parallel algorithm. Let us analyze them one by one: *Time* is probably the most used measure in complexity theory. On the other hand, in the real-time area, time is in most cases predetermined by the existence of deadlines imposed to the computation or by other similar time constraints. Admittedly, there are classes of real-time algorithms for which running time actually makes sense as a measure of performance. However, time is no longer a universal performance measure in the real-time environment. Things are different as far as *space* is concerned though. Indeed, space as a performance measure bears the same significance in a real-time environment as it does in classical complexity theory. We therefore introduce the corresponding classes \( rt - \text{SPACE}(f) \) (corresponding to \( \text{SPACE}(f) \) in classical complexity theory).

A third measure of interest is the *number of processors*. Parallel real-time algorithms have been shown to make up for the limited time that is available [6, 14]. Thus, we consider the classes \( rt - \text{PROC}(f) \). In addition, \( rt - \text{PROC}(f) \) may be different when the PRAM is the model of choice, as opposed to, say, some interconnection network [2]. Given a model of parallel computation \( M \), we denote the corresponding class by \( rt - \text{PROC}^M(f) \), with the superscript often omitted when either the model is understood from the context, or the class is invariant to the model. We write \( rt - \text{PROC}(c) \) (or \( rt - \text{SPACE}(c) \), etc.) instead of \( rt - \text{PROC}(f) \), whenever \( f(x) = c \) for all \( x \in \mathbb{N} \).

**Definition 1** Given a total function \( f : \mathbb{N} \rightarrow \mathbb{N} \), and some model of parallel computation \( M \), the class \( rt - \text{SPACE}^M(f) \) consists in exactly all the well-behaved timed ω-languages \( L \) for which there exists a real-time algorithm running on \( M \) that accepts \( L \) and uses no more than \( f(n) \) space, where \( n \) is the size of the current input. Analogously, the class \( rt - \text{PROC}^M(f) \) includes exactly all the well-behaved timed ω-languages \( L \) for which there exists a real-time algorithm running on \( M \) that accepts \( L \) and uses no more than \( f(n) \) processors on any input of size \( n \). By convention, the class \( rt - \text{PROC}^M(1) \) (that is, the class of sequential real-time algorithms) is invariant with \( M \).
How to measure The notion of input size is not explained in Definition 1. In the classical theory, the input size is the (total) length of the input. However, using such a definition, all well-behaved timed $\omega$-words have length $\omega$. A new notion of input size should be therefore developed.

Take the domain of real-time database systems. Here, the most time consuming operation is answering queries that appear as input. That is, at any moment when some new input arrives, this input consists in the $n$ symbols that encode a query. Motivated by this, we propose the following definition for input size: The size of some $\omega$-word $w$ is given by the largest bundle that arrives as input at the same time. Indeed, such a definition makes sense only when the real-time algorithms manifest the pseudo-on-line property (when they process input data in bundles, without knowledge of future input), but it would appear that this is a common feature of such algorithms [9].

Definition 2 Let $w = (\sigma, \tau)$ be some timed $\omega$-word, \(\tau = \tau_1 \tau_2 \tau_3 \ldots\), \(\sigma = \sigma_1 \sigma_2 \sigma_3 \ldots\). For $i_0 = 0$ and any $j > 0$, let $s_j = \sigma_{j-1} + 1 \sigma_{j-1} + 2 \ldots \sigma_j$, such that (a) $\tau_{j-1} + 1 = \tau_{j-1} + 2 = \cdots = \tau_j$, and (b) $\tau_{j+1} \neq \tau_j$. Then, the size $|w|$ of $w$ is $|w| = \max_{j \geq 0} |s_j|$.

4 The Hierarchy $rt - \text{PROC}(f)$

Given some arbitrary word $w$ of length $n$, we shall index it starting from 0 (however, both the symbol and time sequences in a timed $\omega$-word are indexed in [9] starting from 1, and we shall keep this convention). For any $0 \leq i \leq j < n$, we denote by $w_{i,j}$ the subword $w_i w_{i+1} \cdots w_j$ of $w$. Throughout Sections 4.1 and 4.2 we use the CRCW PRAM [2] as our computational model.

4.1 Two Processors

We construct in what follows a timed $\omega$-language $L_1$ which is accepted by a two-processor algorithm, but cannot be accepted by a sequential algorithm.

Fix $r$ and $p$, $r > 2p$, and \(\Sigma = \{a, b, +, -, \} \). Let $L_0 = \{ (\sigma, \tau) | \sigma \in \{ a, b \}^*, \tau_i = 0 \}$ for all $1 \leq i \leq r \}$. A word in $L_0$ represents an initial value that will be modified as time passes. Such a modification is given by $L_t = \{ (\sigma, \tau) | \tau_j = j, 1 \leq j \leq p + 1, \sigma_1 \in \{ +, - \}, \sigma_{2..j} \in \{ a, b \}^{j-1}, \tau_i = t \}$ for all $1 \leq i \leq j \}$. A word in $L_t$ denotes a change arriving at time $t$: the first symbol is $+$ or $-$, followed by at most $p$ $a$’s and/or $b$’s. The semantics will become clear shortly. Let $L_t = \prod_{t \geq 0} L_{ct}$, for a given positive $c$. $L_1$ will be constructed as a subset of $L_0 L_t$. We need some new concepts to precisely define $L_1$.

Let $w \in \{ a, b \}^*$ and $u = u_0 u' \in \{ +, - \}$ and $d \in \{ a, b \}^l$, $j \leq p$. We define the insertion modulo $r$ at point $i$ of $u$ in $w$, $0 \leq i < r$, as a function $\text{ins}_t$ that receives $w$, $u$, and $i$, and returns a new word $w$ and a new $i$ as follows: Let $i' = i + p$ if $u_0 = +$ and $i' = i - p$ otherwise. Then, $\text{ins}_t(w, u, i) = (w', i' \mod r)$, where $w'$ is computed
as follows (denoting the reversal of some word \( x \)): (a) If \( i' < 0 \) (and thus \( u_0 = - \)), let \( i'' = i' \mod r \); then, \( w' = w_{i_0...i'}w_{i'+1...i''-1}w_{i''+1...i-1} \). (b) Analogously, if \( i' > r - 1 \) (and thus \( u_0 = + \)), then \( w' = w_{i'-i...i-1}w_{i'+1...i''-1}w_{i''+1...i-1} \). (c) Otherwise (that is, when \( 0 \leq i' \leq r - 1 \)), let \( i_1 = \min(i, i') \), \( i_2 = \max(i, i') \), and \( x = u' \) if \( u_0 = + \) and \( x = u' \) otherwise; then, \( w' = w_{i_0...i_1-1}w_{i_2+1...i'-1} \). Denote \( \text{ins}_r(w, u, i) \) by \( (w, i) \oplus u \), and let \( \oplus \) be a left-associative operator. Then, for some integers \( \alpha \) and \( \beta \), \( 1 \leq \alpha \leq \beta \), for appropriate words \( w \), \( u^\alpha \), \( u^{\alpha+1} \), \ldots, \( u^\beta \), and for some \( i \), \( 0 \leq i \leq r - 1 \), we define \( (w, i) \bigoplus_{j=\alpha}^{\beta} u^j = (w, i) \oplus u^\alpha \oplus u^{\alpha+1} \oplus \cdots \oplus u^\beta \).

**Figure 1:** Pursuit and evasion: insertion modulo \( r \) (a, b); and acceptable insertion zone (c, d).

---

**AN INTUITIVE ASIDE.**

The intuition behind \( \text{ins}_r \) (also suggested by its name) is simple: Picture the word \( w \) as a circle, in which \( w_0 \) is adjacent to the right to \( w_{r-1} \). Then, \( u \) replaces \( j \) consecutive symbols in the "circle" \( w \), starting from \( w_i \), and going either to the left or to the right, depending on \( u_0 \) (+ for right). \( u \) models the moves of the *pursuee* over the "circle" \( w \). The pursuee has a topmost velocity of \( p/r \)-th of the circle’s circumference per time unit. The algorithm that accepts \( L_1 \) (the pursuer) will have to match the moves of the pursuee, and its “velocity” is proportional to the speed of the processor(s) used.

Specifically, consider the example in Figures (1.a) and (1.b), where \( r = 8 \).
and \( p = 3 \). Initially, \( w = bbbbbbbb \), and the insertion point is \( i = 1 \). \( w \) is represented in the figure by locations on a circle, labeled with their indices (inside the circle), and with the values stored there (outside). Figure (1.a) shows the insertion of \( a = -a_1 a_2 a_3 \). The pursuee moves to the left, rewriting the symbols at indices 1, 0, and 7, in this order. After the insertion, \( i = 7 \) and \( w = a_2 a_1 b b b b a_3 \). Then, let \( u = +a_4 a_5 \). The indices that are modified are 7 and 0, as shown in Figure (1.b). The result is \( i = 0 \) and \( w = a_5 a_1 b b b b a_4 \).

\( \otimes \) is for \( \oplus \) as \( \Sigma \) is for + in arithmetic. For example, refer to Figure (1.b), which illustrates the result of \( (b b b b b b b b b b, 1) \otimes \sum_{j=1}^7 w^j \), where \( w^1 = -a_1 a_2 a_3 \), and \( u^2 = +a_4 a_5 \). The result is \( (a_5 a_1 b b b b a_4, 0) \).

Consider now \( w \in L_n L_{\eta}^q \), \( w = w^0 \prod_{i=0}^s w_i \), with \( w^0 \in L_0 \), and \( w_i \in L_{\eta_i}, i > 0 \). For some \( t \) and some \( i_0 \), \( 0 \leq i_0 \leq r - 1 \), let \( s(w,t) = (a^{i_0}, i_0) \otimes_{i \leq r} \sigma_i \), where \( \sigma_i = \text{detime} (w_i) \), \( i \geq 0 \).

Let \( A \) be an algorithm that receives \( w \) and uses \( \pi \) processors, \( \pi \geq 1 \) (\( A \) is sequential if \( \pi = 1 \) and parallel otherwise). \( A \) may inspect (i.e., read from memory) the symbols stored at some indices in \( s(w,t) \). Many processors may inspect different indices in parallel. For each processor \( q \), let \( t_q^i \) be the most recent index inspected by processor \( q \) up to time \( t \). If some processor inspects no symbols from \( s(w,t) \), then \( t_q^i = -1 \). Let \( I_q^t \) be the “history” of inspected symbols up to time \( t \), i.e., \( I_q^t = \bigcup_{j=0}^t t_q^j \setminus \{-1\} \). Assume that \( A \) does not inspect any symbol whatsoever from \( s(w,t) \) (e.g., \( A \) doesn’t even bother to maintain \( s(w,t) \) in memory). Then, for any \( t \), \( t_q^0 = -1 \) and thus \( I_q^t = \emptyset \), \( 1 \leq q \leq \pi \). Let \( lo = \min_{1 \leq q \leq \pi}(t_q^1) \), \( hi = \max_{1 \leq q \leq \pi}(t_q^1) \), and \( I = \bigcup_{1 \leq q \leq \pi} I_q^1 \). Then, we define \( z(w,t) \), the acceptable insertion zone at time \( t \), as follows:

\[
z(w,t) = \begin{cases} 
\{i|0 \leq i < r\} & \text{if } lo = -1, \\
\{i|0 \leq i < r, i \neq lo\} & \text{if } lo \neq -1 \text{ and there exists } j \notin I, j > hi \text{ or } j < lo, \\
\{i|lo \leq i \leq hi\} & \text{otherwise.}
\end{cases}
\tag{1}
\]

When the area delimited by the latest inspected indices has been seen, then this area is excluded from \( z(w,t) \). Otherwise, \( z(w,t) \) contains all the indices, except the smallest (if any) positive \( t_q^i \).

**Observation 1** If \( \pi = 1 \) and at least one index has been inspected, then \( |z(w,t)| = r - 1 \) for any \( t > 0 \). Generally, if \( \pi = 1 \), then \( z(w,t) \geq r - 1 \).

---

**AN INTUITIVE ASIDE.**

In Figure (1.c) and (1.d), the acceptable insertion zone is denoted by white bullets. Assuming two processors (or pursuers), Figure (1.c) represents the acceptable zone at the moment in which the two processors inspect

\[\text{Recall that, for some timed } \omega \text{-word } w = (\sigma, \tau), \text{ detime}(w) = \sigma, \text{ as stated in Section 2.}\]
indices 1 and 6, provided that, say, processor $p_1$ started from index 0 and inspected only indices 0 and 1, and processor $p_2$ inspected only indices 7 and 6, in this order. When one processor is available, the acceptable insertion zone is the whole circle but one index (Figure (1.d)).

All the indices outside the acceptable insertion zone must have been inspected: Take Figure (1c). If index 7 is not inspected at all, the acceptable insertion zone is identical to the one in Figure (1.d) by Relation (1). That is, the pursuer(s) must walk over the circumference; if they jump all over the place, then the pursuee is allowed to change almost any index. As we shall see, this makes it uncatchable.

For $w \in L_a L_u$, let $z_i(w)$ be the set of indices whose values are modified by $w' \in L_{ci}$. $L_1 = \{ w \in L_a L_u | z_i(w) \subseteq z(w, ci), i > 0, \text{ and there exists } t > 0 \text{ and } i_0, 0 \leq i_0 < r, \text{ s.t. } |s(w, t)| = |s(w, t)|_a \}$.

**Lemma 1** $L_1$ is a well-behaved timed $\omega$-language.

**Proof Sketch.** Monotonicity and progress follow from Theorem 1 and from the bounded length of $L_{ci}$.

**Lemma 2** There exists no sequential deterministic real-time algorithm that accepts $L_1$.

**Proof Sketch.** Assume such an algorithm $A$ exists, and (a) $A$ has instant access to $s(w, t)$; and (b) $A$ is able to decide whether $|x|_a = |x|_b$ in $|x|$ steps. The nonexistence of $A$ implies the nonexistence of an algorithm accepting $L_1$. If the processor used by $A$ has the ability to inspect $c_1 > 0$ symbols per time unit, choose $p = c \times c_1 + 1$. $A$ can inspect at most $p - 1$ symbols in time $c$; i.e., at any time $t$ the value of at least one symbol in $s(w, t)$ is unknown to $A$. However, there exists a word $w$ such that, at each time $ci$, exactly $p$ symbols are inserted in $s(w, t)$. Without loss of generality, let the insertion point be $j = \lfloor r/2 \rfloor$. By Observation 1, $z(w, t)$ contains exactly all the indices in $s(w, t)$, except one (denoted by $z_i$). If $0 \leq z_i < j$, then $w^{i+1} = +x, |x| = p$. Otherwise, $w^{i+1} = -x$.

**Observation 2** For some 2-processor algorithm $A'$ and some input $w$ ($w = w^0 \prod_{i \geq 0} w^i, w^0 \in L_a$ and $w^i \in L_{ci}, i > 0$), and under a judiciously chosen order of inspection of $s(w, t)$, (a) $|z(w, t)|$ is decreasing with respect to $t$, and (b) for any $x \geq 0$, there exists a finite $t$ such that $|z(w, t)| < x$.

**Proof Sketch.** Let processors $p_1, p_2$ used by $A'$ be able to inspect $\varepsilon c_1$ symbols per time unit each, where $c_1$ is as in the proof of Lemma 2, and $\varepsilon$ is arbitrarily close to 0. Let $t_0 = 0$, $t_1 = 1$. If $p_1$ inspects at $t_1 + 1$ a new index in $s(w, t)$, let it be $t_1^{i_1} + 1$. Analogously, if $p_2$ inspects at $t_2 + 1$ a new index, let it be $(t_2^2 - 1) \mod r$. 


If neither $p_1$ nor $p_2$ inspected any index at time $t + 1$, then $|z(w, t)| = |z(w, t + 1)|$; otherwise, $|z(w, t)| > |z(w, t + 1)|$. $A'$ updates $s(w, t)$ in finite time by Lemma 1. □

**Lemma 3** There exists a 2-processor PRAM deterministic real-time algorithm that accepts $L_1$ and that uses arbitrarily slow processors.

Lemma 3 is immediate from Observation 2. Lemmas 1, 2, and 3 imply:

**Theorem 2** $rt - \text{PROC}(1) \subset rt - \text{PROC}^\text{PRAM}(2)$ (strict inclusion).

Theorem 2 states that a parallel real-time algorithm is more powerful than a sequential one, even if the speed of the processors that are used by the former is arbitrarily smaller than the speed of the unique processor used by the latter. To our knowledge, this is the first result of this nature to date. In fact, we can improve on the result stated in Theorem 2.

### 4.2 $n$ Processors

A form of Theorem 2 holds for any number of processors $n$, $n > 1$; i.e., not only parallel real-time implementations are more powerful than sequential ones, but they also form an infinite hierarchy with respect to the number of processors used: Given any number of processors available to a parallel real-time algorithm, there are problems that are not solvable by that algorithm, but that are solvable if the number of available processors is increased, even if each processor in the new (augmented) set is (arbitrarily) slower than each processor in the initial set. To show this, we develop a language $L_k$ similar to $L_1$, that extends the “circle” expressed by $L_1$ to $k$ dimensions. Fix $k > 1$, $p > 0$, $r > 2p$, $r' = kr$. Put $L'_0 = \{(\sigma, \tau) | \sigma \in \{a, b \}^{r'}, \tau_i = 0 \text{ for all } 1 \leq i \leq r' \}$.

Let $N_k = \{\text{enc}(i) | 1 \leq i \leq k\}$, where $\text{enc}$ is a suitable encoding function from $\mathbb{N}$ to $\{1\}^*$. Let $i \notin \Sigma$. It is assumed that $|\text{enc}(j)| \leq j$ for any $j \in \mathbb{N}$, and that $\text{enc}^{-1}$ is defined everywhere and computable in finite time (these properties clearly hold for any reasonable encoding function). Define $L^0_i = \{(\sigma, \tau) | \sigma \in N_k, \tau_i = t \text{ for all } 1 \leq i \leq |\sigma| \}$. Then, the multi-dimensional version of $L_i$ is $L'_i = L^0_i L_4$ (a word in $L'_i$ also provides the “dimension,” from 1 to $k$, along which the insertion takes place). Let $L'_u = \prod_{i > 0} L'_i$, for a given constant $c > 0$. $L_k$ will be a subset of $L'_u$.

For $w \in \{a, b\}^r$, let $w = w(1)w(2) \ldots w(k)$, $|w(i)| = r$, $1 \leq i \leq k$ ($w(i)$ is a segment of $w$), and $u = u'u''$, $u' \in N_k$ and $u'' \in \Sigma^l$, $1 \leq j \leq p + 1$, $u''_1 \in \{+,-\}$, $u''_1 \ldots p \in \{a, b\}^{r - 1}$. For $0 \leq i < r$, define $(w, i) \otimes u = \left(\prod_{j = 1}^{i - 1} w(j)\right) ((w(d), i) \oplus u') \left(\prod_{j = d + 1}^{k} w(j)\right)$, where $d = \text{enc}^{-1}(u')$. In other words, the word to be inserted contains two components, one of them ($u'$) encoding a number, and the other one ($u''$) denoting the actual word that is to be inserted; the left-associative operator $\otimes$ inserts (modulo $r$) $u''$ into that segment of $w$ which is given by $u'$. $\otimes$ is defined analogously to $\oplus$. 
For some $w \in L'_n L'_w$ ($w = w^0 \prod_{i > 0} w^i$, with $w^0 \in L'_n$, and $w^i \in L'_w$), and for some $i_0$, $0 \leq i_0 \leq r - 1$, let $s'(w, t) = (\sigma^0, i_0) \bigotimes_{i, t \leq i} \sigma^i$, where $\sigma^i = \text{detime}(w^i)$, $i \geq 0$. One should note that $s'(w, t)$ is a generalization of $s(w, t)$ defined in Section 4.1. As a consequence, the concept of acceptable insertion zone can be naturally extended. Indeed, consider the same algorithm $A$ that receives some $w / \mathcal{B} \mathcal{E}$ as input and uses $\pi$ processors. Then, for some $t \geq 0$, define $z'(w, t) = z(w(j), t)$, $1 \leq j \leq k$, with $z(w(j), t)$ defined as in Relation (1), except for the following change: if, at time $t$, some processor inspects an index outside $s(w(j), t)$, then $v_i(t) = -1$ and $V_i(t) = 0$. Finally, let $z'(w, t) = \bigcup_{j=1}^k z'(w, t)$, and call $z'(w, t)$ the acceptable insertion zone at time $t$.

**AN INTUITIVE ASIDE.**

The $k$-dimensional geometric version is an extension of the one-dimensional one. We refer to Figure 2. Each dimension is represented by a circle whose circumference has length $r$. There are $k$ such circles. Each collection of $k$ identical indices (one on each circle) is connected by a special path (there are $r$ such paths, represented by thinner lines in Figure 2). These paths can be used by the pursuee at no cost. However, the pursuers are too bulky to take such narrow paths (more precisely, once a pursuer uses such a path, it looses the advantage gained by the existence of the acceptable insertion zone, similarly to the case of jumping pursuers in the one-dimensional case — see Figure (1.c) and (1.d)).

With $\hat{z}(w)$ the set of indices whose values are modified by the subword $w^i \in L_{ci}$ of $w$, $L_k = \{ w \in L'_n L'_w \}$ for $i > 0$, $\hat{z}(w) \subseteq \hat{z}(w, ci)$, and there exists $t > 0$ and $i_0$, $0 \leq i_0 < r$, s. t. $|s'(w, t)| = |s'(w, t)|$. In Lemma 4, (a) is a trivial generalization of Lemma 1, (b) is proved by induction over $n$, using Lemma 2; if we allocate two processors for each $s'(w, t)(j)$, $1 \leq j \leq n$ we can use Lemma 3 in order to
prove (c). Theorem 3, stating the infinite hierarchy \( rt \prec \text{PROC}^{\text{PRAM}}(f) \), follows from Lemma 4.

**Lemma 4** (a) \( L_k \) is a well-behaved timed \( \omega \)-language for any \( k > 1 \). (b) There exists no \( (2n-1) \)-processor PRAM deterministic real-time algorithm that accepts \( L_n \), \( n \geq 1 \). (c) There exists a \( 2n \)-processor PRAM deterministic real-time algorithm that accepts \( L_n \) and that uses arbitrarily slow processors, \( n \geq 1 \).

**Theorem 3** For any \( n \in \mathbb{N}, n \geq 1 \), \( rt \prec \text{PROC}^{\text{PRAM}}(2n-1) \subset rt \prec \text{PROC}^{\text{PRAM}}(2n) \) (strict inclusion).

### 4.3 Other Parallel Models of Computation

One may wonder whether Theorem 3 holds for other models of computation besides the PRAM. A model that allows a straightforward implementation, as opposed to the PRAM, is the *bounded-degree network* (BDN) [2], where communication between processors is achieved using a sparse interconnection network of fixed degree. However, it is well-known that even the most powerful version of the PRAM (namely, the CRCW PRAM [2]) can be simulated on a BDN with bounded slowdown and bounded memory blowup. Specifically, there exists a simulation [11] for which the slowdown is \( O(\log^2 n/\log \log n) \), and the memory blowup is \( O(\log m/\log \log m) \), where \( n \) is the number of processing elements, and \( m \) is the amount of memory used by the PRAM.

However, a bounded slowdown does not affect the result in Theorem 3, since this result is invariant to the speed of the processors involved. Furthermore, the PRAM algorithm uses a finite amount of memory; thus, a bounded memory blowup results in a finite amount of memory as well for the BDN that simulates the PRAM algorithm. In addition, given that the BDN allows for an immediate physical implementation, we make the following (arguable, but nevertheless often encountered) assumption: The BDN is the most elementary model of parallel computation. With this assumption in mind, the following result is an immediate corollary of Theorem 3:

**Theorem 4** Given any model of parallel computation \( M \), and for any \( n \in \mathbb{N}, n \geq 1 \), \( rt \prec \text{PROC}^M(2n-1) \subset rt \prec \text{PROC}^M(2n) \) (strict inclusion).

That is, we have not only an infinite hierarchy \( rt \prec \text{PROC}^{\text{PRAM}}(f) \), but such a result holds for \( rt \prec \text{PROC}^M(f) \) as well, for any model of parallel computation \( M \).

### 5 Conclusions

In one of our previous paper [9], we suggested as an interesting research direction the study of a realistic computational complexity theory for parallel real-time
systems, that is, a theory based on a realistic model of real-time computations, which can be easily translated into practice. The concept of timed ω-languages was proposed in [9] as a possible foundation for this pursuit. We continue this idea here. We started by defining complexity classes for the real-time domain. We believe that Definition 1 captures the intuitive notion of resource (processors, storage space) bounds for real-time parallel algorithms. We also believe that these resources are the most important for the domain. However, Section 3 offers the basis for the development of other complexity classes, should they prove to be useful.

Besides defining the basis for our theory, we also proved what we believe to be an important result. Indeed, Theorem 4 shows that the hierarchy of real-time algorithms with respect to the number of processors is infinite, and that such a result is invariant to the model of parallel computation involved, and independent of the characteristics (that is, speed) of the particular processors used by the algorithms. To our knowledge, this is the first time such a result is proved. From a practical point of view, Theorem 4 emphasizes the need for looking into parallel implementations, since this theorem shows that parallelism may add power, in a more general sense than mere speed, to a real-time application.

The languages $L_k$, $k \geq 1$, faithfully model the geometrical variant of the problem. We chose this direction in order to preserve the clear and intuitive support provided by the geometrical case. However, the notion of insertion point (that moves after each insertion) is not necessary. It is immediate that the results in this paper hold even if the input is allowed to change (any number of) arbitrary indices within the acceptable insertion zone at any time $c_i$, $i > 0$.

It should be noted that, even if the algorithms developed in this paper do not exhibit explicit deadlines, they are nonetheless real-time algorithms. Indeed, as opposed to on-line algorithms [12] (that have no time constraints whatsoever), the input arrives in real-time here. Such an input arrival implicitly imposes a sequence of deadlines (in a certain sense) on the algorithm itself [7, 17] (for example, many algorithms with real-time input constraints should process the current input before another datum arrives in order to successfully handle the (real-time) input arrival [17]; as another example, although no explicit deadlines are present in the case of $d$-algorithms, it is shown in [7] that the real-time arrival law for the input actually imposes a deadline on the length of the computation of any $d$-algorithm). Besides, practical domains such as real-time databases and industrial applications include input into the factors determining the real-time qualifier [15, 16, 17].

A note on the differences and similarities between timed ω-languages (that is, real-time algorithms) and classical formal languages (that is, classical algorithms) is also in order. On one hand, it is immediate that formal languages are particular cases of timed ω-languages. Indeed, save for the time sequence, any word is a timed ω-word. If one relies on the semantics of the time sequence, one can add the time sequence $00\ldots0$ to a classical word and obtain the corresponding timed ω-word. However, none of the timed ω-words obtained in this manner is
well-behaved. We have thus a crisp delimitation between real-time and classical algorithms, while keeping the formalisms as unified as possible.

As we have mentioned in Section 1, this paper establishes theoretical results. Still, such results offer a good motivation to the pursuit of a real-time parallel complexity theory. Besides, they, together with other similar ones, may be useful when descriptions based on timed $\omega$-languages for real world problems are developed. In particular, we are interested in two research directions: First, and from a theoretical point of view, we intend to investigate computations exhibiting deadlines (since it appears that in practice deadlines are the most encountered type of real-time constraints), and try to relate classical complexity classes with real-time ones. For example, we wonder whether there exists some well-defined complexity class(es) that include those problems solvable in parallel in the presence of deadlines. Secondly, we already modeled the recognition problem for real-time database systems, and the routing problem in ad hoc networks [10]. We are now interested in developing complexity theoretic characterizations for those problems, based on the developed models.

References


**Stefan D. Bruda and Selim G. Akl** are with the Department of Computing and Information Science, Queen’s University, Kingston, Ontario, K7L 3N6 Canada. Email: {bruda,akl}@cs.queensu.ca.
Group Mutual Exclusion in Token Rings *

SÉBASTIEN CANTARELL
LRI/CNRS, Université de Paris-Sud, France

AJAY K. DATTA
University of Nevada Las Vegas, USA

FRANCK PETIT
LaRIA, Université de Picardie Jules Verne, France

VINCENT VILLAIN
LaRIA, Université de Picardie Jules Verne, France

Abstract
The group mutual exclusion (GME) problem was introduced by Joung [6]. The GME solution allows $n$ processes to share $m$ mutually exclusive resources. We first present a group mutual exclusion algorithm (Algorithm $GME$) for anonymous token rings. The space requirement and the size of messages of this algorithm depend only on the number of shared resources ($O(\log m)$ bits). So, the proposed algorithm solves the problem suggested in [7], which is to obtain a solution using messages of bounded size. All costs related to the time depend on $n$. We then present two variations of Algorithm $GME$. We design the second algorithm (Algorithm $mgME$) such that its cost depends mainly on the $m$ instead of $n$. The third algorithm (Algorithm $nmgME$) is a general algorithm which takes advantage of the lowest value between $n$ and $m$.

Keywords
distributed algorithms, group mutual exclusion, mutual exclusion

1 Introduction
The mutual exclusion and concurrency are among the fundamental problems of distributed systems. The mutual exclusion ensures an exclusive access to a shared resource (also known as sessions [8]) among a set of processes. The concurrency allows some processes to share a resource.

* This work was supported in part by the Pole of Modelization of Picardie.
The group mutual exclusion (GME) problem was recently introduced by Joung [6]. This problem deals with both mutual exclusion and concurrency problem. The GME solution allows \( n \) processes to share \( m \) mutually exclusive resources. At any time, some processes share a particular resource. But, if a process requests to access a resource different from the currently used resource, then the process cannot access the requested resource at that time. However, if a process requests the resource being currently used, then the process is allowed to share the resource with other processes. There is no limit on the number of processes which can use a resource concurrently.

An interesting application of the group mutual exclusion is presented in [6]. Consider large data sets stored in a secondary memory. A set of processes accesses the data sets through a server. The server can be a CD jukebox. Using a purely mutual exclusion protocol, the server needs to repeatedly load and unload the data sets (e.g., the CDs) from the secondary memory to process the requests. An efficient GME protocol would allow multiple processes to read the currently loaded data set (a CD) simultaneously, while forcing the processes requesting a different data set (another CD) to wait. Furthermore, while a data set remains loaded, it remains accessible by the processes requesting the set until some other data set is requested by some process.

An efficient GME solution could also help improve the quality of services (QoS) of an Internet server. The GME protocol could be used to group different requests for the same service, and thereby, minimize the memory swapping.

**Related Work.** The GME problem [6] is a generalization of mutual exclusion [3, 10] and readers/writers [2] problem. It is also related to several widely studied synchronization problems such as dining philosophers [4], drinking philosophers [1], and \( k \)-exclusion [5]. Refer to [8] for discussion on the use of GME to solve these problems.

In [6], Joung proposed a solution to the group mutual exclusion problem for the shared-memory model. The solution in [8] also runs in the shared memory systems. In [6], the author introduced a new performance metric, called the context-switch complexity, different from the classical time complexity. The time complexity measures the total number of critical section accesses by processes while a particular process is waiting to access the critical section. The context-switch complexity indicates the number of sessions which can be opened while a particular process is waiting. This new measure takes in account the accesses overlapped due to the concurrency. Keane and Moir [8] introduced another measure related to the access time, called the contention-free complexity, which shows the worst-case time needed by a process to execute its entry and exit sections in the absence of contention. The (maximum) degree of concurrency is another specific performance introduced in [6]. The degree of concurrency captures the number of processes that can concurrently access an opened session while a particular process is waiting for a session. Higher degree of concurrency
implies better resource utilization.

Solutions for the message-passing model were first presented in [7, 11]. The algorithms of [7] considered fully connected networks. In [11], the authors presented two algorithms for unidirectional rings. The number of messages generated by each access to the critical section in both algorithms of [11] is $\Theta(n^2)$ messages. The time complexity and the degree of concurrency is $O(n^2)$ for both algorithms in [11]. The context-switch complexity of the two algorithms in [11] are $O(n^2)$ and $O(\min(n, m))$. Although the contention-free complexity is not discussed in [11], it is easy to observe that in both algorithms, the contention-free complexity is equal to the message complexity, i.e., $\Theta(n)$.

All GME algorithms to date [6, 7, 8, 11] consider networks where processes have unique id’s. Another common characteristic of these algorithms is that they all maintain some data structures to store process id’s and resource id’s, and use these data structures to control access to $m$ resources by $n$ processes. In the message passing solutions of [7, 11], these data structures are locally maintained by the processes. Moreover, the solutions of [7, 11] require to maintain locally a logical clock [9] which cannot be bounded. The messages must carry the process id’s, the resource id’s, and the logical clock value. Therefore, the processes must have an infinite memory, and the message size cannot be bounded.

**Contributions.** In this paper, we first present a simple group mutual exclusion algorithm (Algorithm $GME$) for unidirectional rings. As in [8], Algorithm $GME$ uses an underlying mutual exclusion protocol, more specifically, a token circulation mechanism (also referred to as token ring in the following). The proposed algorithm works on anonymous networks. However, the token circulation must be initiated by a unique process.

Algorithm $GME$ does not maintain any special data structure to implement any queue. The space requirement of processes depends only on the number of shared resources, and is $4 \times [\log(m + 1)] + 2$ bits. The size of messages is $2 \times [\log(m + 1)]$ bits. So, the proposed algorithm solves the problem suggested in [7], which is to obtain a solution using messages of bounded size.

Each access to the critical section generates $0$ to $O(n)$ messages. As in [11], the time complexity depends on the message complexity, and the contention-free complexity depends on the network architecture. The time complexity is $O(n^2)$, but the contention-free complexity is $O(n)$ ($\Theta(n)$ in [11]) because it is $0$ in the best case and $n$ in the worst case. The context-switch complexity of Algorithm $GME$ is $O(n)$. The degree of concurrency cannot be bounded. This shows that Algorithm $GME$ provides the best possible resource utilization.

We then present two variations of Algorithm $GME$. These two algorithms perform better than Algorithm $GME$ in some situations. The cost of Algorithm $GME$ depends only on $n$. So, this algorithm performs its best in systems with $n < m$. We design the second algorithm (Algorithm $mGME$) such that its cost depends mainly on the $m$ instead of $n$. So, Algorithm $mGME$ is suitable for
systems where \( m < n \). The time complexity and the context-switch complexity of Algorithm \( mGME \) is \( O(nm) \) and \( O(m) \), respectively. Moreover, the algorithm achieves this performance without any extra information in the token. So, the message size complexity remains at \( O(\log m) \) bits.

The third algorithm (Algorithm \( nmGME \)) is a general algorithm which takes advantage of the lowest value between \( n \) and \( m \). The context-switch complexity of Algorithm \( nmGME \) is \( O(\min(n,m)) \), while the time complexity is \( O(n \cdot \min(n,m)) \). However, Algorithm \( nmGME \) requires \( O(\log \min(n,m)) \) extra bits in the token and in the space requirement.

**Outline of the Paper.** The rest of the paper is organized as follows. In Section 2, we describe the distributed systems. In the same section, we give a formal statement of both the problem solved and the different complexity metrics considered in this paper. In Section 3, we present Algorithm \( GME \), followed by the proof outline and some discussion of the complexity issues. Both variations (Algorithms \( mGME \) and \( nmGME \)) are presented in Section 4. Finally, we make some concluding remarks in Section 5.

### 2 Preliminaries

**Distributed Systems.** The distributed system we consider in this paper consists of \( n \) processes (0, 1, \ldots, \( n - 1 \)) with no identity. The numbers 0, 1, \ldots, \( n - 1 \) are used only to simplify the presentation. All operations on the process numbers are assumed to be modulo \( n \). Processes communicate only by message passing and are arranged in a ring topology such that each process \( p \) can only send messages to process \( p + 1 \). The message delivery time is arbitrary (i.e., finite but unbounded).

**GME Problem.** We assume that the processes cycle through a non-critical section, an entry section, a critical section, and an exit section. The processes can access a “session” only within a critical section. We assume that the processes execute their critical section in finite (but unpredictable) time. Every time a process \( p \) moves from its non-critical section to the entry section, \( p \) non-deterministically chooses a session \( s_p \) from \( \{1, \ldots, m\} \). This means that \( p \) requests access to Session \( s_p \). The GME problem is to design a protocol (for the entry and exit sections) so that the following properties are true in every execution:

- **Mutual Exclusion:** If two distinct processes, \( p \) and \( q \), are executing their critical section simultaneously, then \( s_p = s_q \).
- **No Lockout:** If a process \( p \) requests to access a session, then \( p \) eventually executes its critical section.
- **Concurrent Entering:** If a process can access to Session \( X \), and no process is requesting for a different session, then all the processes requesting for Session \( X \) can access to Session \( X \) concurrently.
The "no lockout" property defined above is similar to the "bounded delay" property in [6, 7, 11]. Note that both definitions do not impose any bound on the delay to access the session—the only requirement is that the delay be finite. So, to maintain the 'eventuality' property, we chose the property name as "no lockout" (also called "no starvation") in the above specification. However, the bounded delay property can be used as a stronger requirement for the Group Mutual Problem as follows: There exists a bound on the number of times that other processes are allowed to access another session after a process has made a request to access its session and before that request is granted. Although the bounded delay property is not required to specify the GME problem (as well as the classical mutual exclusion problem), all the deterministic solutions in the literature do satisfy this property. One of the main issues in designing such algorithms is to attain the minimum delay.

Complexity Metrics. We will evaluate our algorithms in terms of communication, space, and time. We will consider some metrics (in the above categories) which are specific to the GME problem [6, 8].

In order to compute the above metrics, we assume that the maximum time needed for a message to move from a process to its successor ($t_{MSG}$) is less than the minimum time needed by a process to execute its critical section ($t_{CS}$), i.e., $t_{MSG} < t_{CS}$. Thus, no process can complete the execution of its critical section while a message is in transit from a process to its successor.

We measure the communication complexities in terms of the message complexity and message size complexity, i.e., the number of messages generated per entry in the critical section and the number of bits per message, respectively. The space complexity is represented by the number of bits used by every process. One of the time metrics is the contention-free complexity which indicates the worst-case time needed by a process to execute its entry and exit sections in the absence of contention.

Before we define the other metrics, we will discuss their purpose. The time complexity measures the number of critical section accesses by processes while a process $q$ is waiting to access a particular session $X$. But, due to the concurrency, the accesses may overlap. So, the time complexity does not truly reflect the elapsed time. The context-switch complexity represents the number of sessions which can be opened while $q$ is waiting. So, the context-switch is a good additional complexity metric to compute the (worst-case) "waiting time". The degree of concurrency captures the number of processes that can concurrently access an opened session while a process $q$ is waiting. Thus, higher degree of concurrency is desirable to achieve higher resource utilization.

We need the term "passage" [6] to define some performance metrics. A passage by a process $p$ through a Session $X$ (denoted by $\langle p, X \rangle$) is an interval $[t_1, t_2]$ (of time) during which Process $p$ executes its critical section. A passage is initiated at $t_1$, and is completed at $t_2$. 
Let $q$ be a process requesting to access Session $X$. Let $T$ be the set of passages initiated by some processes $p$ ($p \neq q$) after $q$ made its request (for Session $X$), and completed before $q$ executes the corresponding passage $(q,X)$. The minimal cover $C$ of $T$ is the minimal subset of $T$ such that every passage of $T$ is initiated and completed during the execution of some passages of $C$. The time complexity is the number of passages in $C$. A round (of passages) $R_Y$ of $T$ is a maximal set of consecutive passages of $T$ which are passages through Session $Y$. The context-switch complexity is the number of rounds of $T$ such that for each round $R_Y$, either $X \neq Y$ or $X = Y$ but $(q,X) \notin R_Y$. The degree of concurrency is defined by the maximum number of passages that can be initiated during a round $R_Y$.

3 Group Mutual Exclusion

3.1 The Basic Algorithm

Let us consider the overall system architecture to describe the message communication. We assume that there exist two layers in the system: the application layer and the GME layer as the lower of the two layers. Such a general structure is well-known. The interface between the two layers is implemented by using two types of messages: Request $Session$ and Grant $Session$. When the application layer needs to access a session, say Session $X$, the process running the application layer sends the message Request $Session$($X$) to the GME layer. Eventually, the GME layer grants the application layer the access to Session $X$ by sending the message Grant $Session$. On completion of its work using Session $X$, the application layer process sends Release $Session$ message to the GME layer.

The GME layer (in the following, also referred to as Algorithm $GME$) is based on a token circulation in rings. The token is used only to open and close a session. Initially, the first process $p$ entering a session $X$ advertises (using the token) to other processes that session $X$ is open. Process $p$ is considered the Leader of the current session. If another process requests for another session, say $Y$, Process $p$ is in charge of closing Session $X$ and opening the next session $Y$. Next, a new Leader for Session $Y$ is chosen. Let us now explain the details of Algorithm $GME$ (Algorithms 3.1 and 3.2).

Algorithm $GME$ consists of three sections. The entry section (Lines 1.01 to 1.06) is executed when a process requests to enter the critical section corresponding to a session. The exit section (Lines 2.01 to 2.07) is executed when a process decides to leave the critical session. The token circulation section (Lines 3.01 to 3.32 and Lines 4.01 to 4.07) is used to manage the opening and closing of sessions. We assume that the group of statements corresponding to any of the above sections is executed atomically. Also, one and exactly one process initiates the algorithm.

Processor $p$ maintains four local variables:
Algorithm 3.1 Variables, Entry and Exit Sections

Variables:
- \( ReqS, CurrentS \in \{ \bot, 1, \ldots, m \} \), initially \( \bot \);
- \( Token\_Held, Leader \) : boolean, initially \( \text{false} \);

Uses:
- function \( InCS() \) : boolean
  /* Return true if in Critical Section, false otherwise */

Predicate:
- \( LExists \equiv (Token.C \neq Token.N) \)

actions:

/**** ENTRY SECTION ****/
1.01 upon receipt of \( \text{Request}\_Session(X) \) from \( ApplicationLayer \)
1.02 \( ReqS := X; \)
1.03 if \( ReqS = CurrentS \)
1.04 then /* Requested session is currently open, enter*/
1.05 send Grant Session to \( ApplicationLayer \)
1.06 endif

/**** EXIT SECTION ****/
2.01 upon receipt of \( \text{Release}\_Session \) from \( ApplicationLayer \)
2.02 \( ReqS := \bot; \)
2.03 if \( Token\_Held \)
2.04 then /* The token must be released */
2.05 \( Token\_Held := \text{false}; \)
2.06 send Token to Successor
2.07 endif

- \( ReqS \): The requested session. \( ReqS = \bot \) means that no session is requested currently.
- \( CurrentS \): The current open session. \( CurrentS = \bot \) implies that no session is currently open.
- \( Token\_Held \): A flag to indicate if \( p \) holds the token.
- \( Leader \): A flag indicating if \( p \) is the leader of the current session.

The message \( Token \) contains two fields: \( Token.C \) and \( Token.N \) (henceforth also denoted by \( (C,N) \)), corresponding to the id of the current and next session, respectively.

We now describe a typical execution of Algorithm \( GME \). Initially, the token is initiated by a process. When the token is intercepted by the first process \( p \) requesting to enter the critical section for Session \( X \), \( p \) becomes the \( Leader \) and sends the token \( (X,\bot) \) to advertise that Session \( X \) is open. While no process requests for another session \( (\neq X) \), \( Token \) circulates in the ring and remains in the same state. Upon receipt of \( Token \), the processes locally store the id of the open session in the variable \( CurrentS \). So, all the processes requesting Session \( X \) may enter the critical section concurrently.

Now, assume another process, say \( q \), requests for Session \( Y \) \( (Y \neq X) \). Since no other process already requested to enter the critical section with another session \( Z \)
Algorithm 3.2 (GME) Group Mutual Exclusion In Token Rings

procedure Token_Processing()

if ~Exists \( \land \) ReqS \( \neq \bot \land \) Token.C \( \in \{ \bot, \text{ReqS} \} \)

then /* Session ReqS is requested and no leader exists */

\( \text{Leader} := \text{true}; \) Token.C := ReqS; CurrentS := ReqS; Token.N := \( \bot \);

send Grant_Session to ApplicationLayer

else

CurrentS := Token.C;

if CurrentS \( \neq \bot \)

then /* A session is currently open */

if ReqS = CurrentS

then /* Requested session is currently open, enter */

send Grant_Session to ApplicationLayer

else

if Token.N = \( \bot \land \) ReqS \( \neq \bot \)

then /* A new session is requested */

Token.N := ReqS

endif

endif

if Token.N \( \neq \bot \land \) Leader

then /* Start closing the current session */

CurrentS := \( \bot \); Token.C := \( \bot \);

Token_Held := InCS()

endif

else /* CurrentS = \( \bot \) */

if Leader

then /* The next session is opened */

CurrentS := Token.N; Token.C := Token.N; Leader := \text{false}

else /* The token must be retained while in critical section */

Token_Held := InCS()

endif

endif

actions:

**** TOKEN CIRCULATION SECTION ****/

upon receipt of Token from Predecessor

Token_Held := False;

Token_Processing();

if ~Token_Held

then /* The token is sent to the next process */

send Token to Successor

endif

(Z \( \neq \) X and Z \( \neq \) Y) and Token.N is still equal to \( \bot \), Process q changes the current value of Token.N from \( \bot \) to Y. This will allow Leader (Process p) of Session X to close the current open session (Session X) and to open the next requested session (Session Y). So, Leader (Process p) of the current open session eventually receives the token \((X,Y)\) and sends the token \((\bot,Y)\). Note that when a process,
say \( r \), receives \( (\perp,Y) \), \( r \) can be in critical section (\text{InCS()} returns true). In this case, \( r \) holds the token (Lines 3.22 or 3.29, \text{TokenHeld} is set to true if \( r \) is in critical section, false otherwise) until \( r \) exits the critical section. When \( r \) exits the critical section, \( r \) releases the token (Lines 2.05 and 2.06). If \( r \) is not in the critical section (\text{InCS()} returns false), \( r \) sends the token to its neighbors (Line 4.06).

Again, \text{Leader} (Process \( p \)) eventually receives the token \( (\perp,Y) \) and then closes the current session. From that point on, Process \( p \) is no longer the leader. Process \( p \) now forwards the token \( <Y,Y> \), meaning that Session \( Y \) is now open. Note that any process may request for any session at any time. So, the new leader may not be the first process who initiated the opening of the session.

Finally, consider the following case. While Session \( X \) is open, two processes \( p \) and \( q \) requested for Session \( Y \) and Session \( Z \), respectively. Although \( q \) made the request before \( p \) did, Session \( Y \) can be initiated before Session \( Z \). The reason is the following: After the request is made by \( q \), the next elected leader can be \( p+1 \), the successor of \( p \) although the corresponding session (Session \( Y \)) was first requested by \( q \). Now, the first process requesting for another session (other than \( Y \)) receiving the token coming from the just elected leader (Process \( p+1 \)) will be able to initiate the next session. So, effectively, if several processes are trying to initiate different sessions, the process nearest from the current session leader would be the first to initiate a new session. The obvious question now is that can a requesting process \( q \) wait forever to initiate a new session? Fortunately, the answer is no. The reason is that the distance between the current leader and \( q \) is finite. So, eventually, \( q \) will be able to initiate its session. The worst scenario is when the successor of \( q \) becomes the next leader.

### 3.2 Proof Outline of Algorithm \( GME \)

By assumption, the system never contains more than one token. From Algorithm 3.2, any process \( p \) can change the value of \text{CurrentSp} iff \( p \) holds the token and can change the value of \text{TokenC} iff \( p \) is a leader. From the above properties, we show that the number of leaders in the rings is at most one. Since a session can be opened or closed by a leader, more than one open session cannot exist at any time. So, the following theorem holds:

**Theorem 1** Algorithm \( GME \) guarantees the mutual exclusion property.

Now, assume that a process \( p \) is the leader of the current session \( X \) and \( q \neq p \) is waiting to enter Session \( Y \) (\( Y \neq X \)) which will be the next open session. Then, the algorithm chooses a process \( q' \) between \( p+1 \) and \( q \) (including \( p+1 \) and \( q \)) as the leader of Session \( Y \). Assume that Session \( Y \) is not the next open session (following Session \( X \)). Then, by applying the same reasoning as above on every process between \( p \) and \( q \), in the worst case, the token can traverse the ring at most \( 2n+1 \) times before \( q \) can enter the critical section. This leads to the following:
Theorem 2 Algorithm GME satisfies the no lockout property.

Finally, assume that a process $p$ can enter Session $X$ while no process is requesting to enter a session $Y \neq X$. Then, $CurrentS_p = X$ ($p$ can enter Session $X$, see Lines 1.05 and 3.05) and the token is equal to $(X, \perp)$ since there is no request for a session other than $X$. So, after having received the token $(X, \perp)$ at least once, all processes requesting to enter Session $X$ can enter Session $X$ concurrently (and any number of times). So, we can claim the following:

Theorem 3 Algorithm GME satisfies the concurrent entering property.

We can claim the final result from Theorems 1, 2, and 3:

Theorem 4 Algorithm GME satisfies the GME specifications (as specified in Section 2).

3.3 Complexity Analysis of Algorithm GME

We consider the following two extreme cases. Assume that a process $p$ requests access to the opened session $X$ and $CurrentS_p$ is set to $X$. In this case, $p$ can access Session $X$ without sending any message (see Lines 1.01 to 1.06). Now, assume that $p$ requests access to a session $X$ which is not the current opened session. In this case, $p$ must wait to access Session $X$. $p$ generates a message carrying its request (by setting Token.N to $X$). This message is eventually received by the current leader which initiates the closing of the current session and then the opening of Session $X$. Process $p$ will be able to execute its critical section when it receives the opening token. So, in the worst case, at most $2n$ messages are necessary for a process $p$ to access its critical section. Hence, the message complexity is in the interval $[0, 2n]$. Note that with the token based algorithms, the message complexity signifies something only when there exists a contention for the critical section entry. Otherwise, even if no request is ever made, an infinite number of messages is sent because the token keeps circulating. However, the algorithm can be easily modified to avoid circulating a token in the absence of contention, i.e., when no new requests are made for a different session.

The message size (or the token size) is fixed: $2\lceil \log(m + 1) \rceil$ bits, where $m$ is the number of sessions. Similarly, the space used by process $p$ can be easily computed by adding the number of bits required by each variable, i.e., $4\lceil \log(m + 1) \rceil + 2$ bits.

We need to consider two cases to compute the contention-free complexity (as we did to compute the message complexity). If $Current_p$ is already set to a session $X$ when $p$ requests access to Session $X$ (Recall that no contention is assumed. So, the session requested by $p$ cannot be other than $X$.), or $Current_p$ is not yet equal to $X$. In the first case, $p$ accesses Session $X$ immediately. In the second case, $p$ accesses Session $X$ after the token traverses the ring twice.
From the proof of Theorem 2, it is clear that at most $n$ rounds of passages can be completed (i.e., $n$ sessions can be opened) while a process $p$ is waiting to access a session. So, the context-switch complexity is $n$ rounds of passages. Moreover, during the opening of each of these $n$ sessions, no process can delay the token progress. By assumption, no process can complete the execution of its critical section while a message is in transit from a process to its successor. Therefore, each component $\alpha$ of the minimal cover $C$ corresponding to the opening of a session contains, in the worst-case (each passage includes the time for a message to move from a process to its successor), $n - 1$ passages. During the closing of each session, in the worst case, the token can be delayed $n - 1$ times, once for each process. While a process holds the token, it can make at most 1 passage. So, again, each component $\beta$ of $C$ corresponding to the closing of a session contains at most $n - 1$ passages (one for each process). So, each component $\alpha \beta$ contains at most $2(n - 1)$ passages. Thus, the time complexity is $2(n - 1)n$ passages, $2(n - 1)$ passages for each of the $n$ rounds.

Following the same reasoning as above, while a process $p$ is waiting for access to a session $X$, the opening component $\alpha$ of $C$ corresponding to a round $R_Y$ (for Session $Y$) contains at most $n - 1$ passages. In the best case, following the process of the token, 1 passage can be completed concurrently with the first passage of $\alpha$, 2 passages can be completed concurrently with the second passage of $\alpha$, ..., $n - 1$ passages can be done concurrently with the $n - 1$th passage. So, the maximum number of passages that can be initiated during $\alpha$ is $n(n - 1)/2$. Again, during the closing of each session, each process $q (\neq p)$ can delay the token while its passage completes. During this time, a process can locally decide if it can execute its critical section ($CurrentS$ contains the requested session number). The process which did not receive the (closing) token can execute an unlimited number of passages concurrently with the passage of $q$. So, the degree of concurrency cannot be bounded. This shows that our algorithm provides the best possible resource utilization.

4 Other Solutions

In this section, we present two variations of Algorithm $GME$. It has been shown in Section 3.3 that for Algorithm $GME$, the context-switch and time complexity is $O(n)$ rounds of passages and $O(n^2)$ passages, respectively. This result is good for systems where $n < m$, i.e., the number of processes participating in the ring ($n$) is less than the number of accessible resources ($m$). The first modification, Algorithm $mGME$, deals with systems where $m < n$. This result is obtained without adding any variable in the processes or fields of the token. All the complexity results of Algorithm $GME$ are preserved, except the context-switch complexity and the time complexity which become $O(m)$ rounds and $O(n \times m)$ passages, respectively. The second modification, Algorithm $nmGME$, is a more
The context-switch complexity of Algorithm $nmGME$ is $O(\min(n,m))$ rounds of passages, while the time complexity is $O(n \times \min(n,m))$ passages. However, Algorithm $nmGME$ requires $O(\log \min(n,m))$ extra bits in the token and in the space requirement.

### 4.1 An $O(m)$ Context-Switch Algorithm

The solution (Algorithm $mGME$) is based on the same mechanism as used in Algorithm $GME$. It uses the same variables and the same code, except that Lines 3.14 to 3.17 of Algorithm 3.2 are replaced by Lines M.01 to M.09 of Algorithm 4.1. The overall idea of Algorithm $mGME$ is that the decision to open the next session is now based on a relative ordering among the sessions. In contrast, the choice made in Algorithm $GME$ is based on the relative position of the processes requesting a session with respect to the current leader process.

Assume that the sessions are numbered from 0 to $m-1$. In order to construct an ordering among the session numbers, a simple but naive solution would be to use the natural ordering on the integers, i.e., $0 < 1 < \ldots < m-1$, and the next opened session would be the lowest (or the highest) requested session number. But, with such a solution, some processes could be locked out by some processes requesting access to sessions with lower (resp., higher) session numbers. Thus, the no lockout property would be violated. This problem can be easily solved by making the ordering circular and start from the position of the current opened session $Token.C$ in the sequence $0 \ldots m-1$. So, if $X = 0$, then the natural order $0 < 1 < \ldots < m-1$ is used. Otherwise, the ordering $X < X+1 < \ldots < m-1 < 0 < 1 < \ldots < X-1$.

Assume that a process $p$ requests access to Session $X$ ($ReqS = X$). When $p$ receives the token, $p$ tests the value of $Token.N$. If $Token.N = \bot$ (no request was made before $p$), then $p$ sets $Token.N$ to $X$ (Line M.03). If $Token.N = Y$ ($y \neq \bot$), then $p$ evaluates the position of $X$ w.r.t. $Y$, relatively to the current opened session $Token.C$. This is implemented by checking the sign of $ReqS \times Token.C$ as the base, i.e., $(ReqS \times Token.C) \times (Token.N \times Token.C)$ (Line M.04). If the sign is positive (both $X$ and $Y$ are either in $[0,Token.C-1]$ or $[Token.C+1,m-1]$), then $Token.N$ is set to $\min(X,Y)$. Otherwise, $Token.N$ is set to $\max(X,Y)$.

It is easy to verify that starting from a configuration where a process $p$ requests for a session, $p$ can receive the token at most $2m+1$ times before $p$ can enter the critical section. So, Algorithm $mGME$ satisfies the no lockout property. Moreover, as for Algorithm $GME$, Algorithm $mGME$ satisfies both the mutual exclusion and the concurrent entering property. Hence:

**Theorem 5** Algorithm $mGME$ satisfies the GME specification (as specified in Section 2).
### Algorithm 4.1 \((mGME)\) A \(O(m)\) GME Algorithm

/* Lines M.01 to M.09 replace Lines 3.14 to 3.17 of Algorithm 3.2 */

M.01 if ReqS ≠ ⊥
M.02 then /* A new session is requested */
M.03 if Token.N = ⊥ then Token.N := ReqS
M.04 else if \((\text{ReqS} = \text{Token.C}) \land (\text{Token.N} = \text{Token.C})\) > 0
M.05 then Token.N := \(\min\{\text{Token.N}, \text{ReqS}\}\)
M.06 else Token.N := \(\max\{\text{Token.N}, \text{ReqS}\}\)
M.07 endif
M.08 endif
M.09 endif

All the complexity results of Algorithm \(mGME\) are similar to the complexity results of Algorithm \(GME\), except the context-switch and time complexity. Following the same analysis as for Algorithm \(GME\), the context-switch and time complexity is \(m\) rounds and \(2n + m\) passages, respectively. It may seem that extra space is necessary to implement the operation of Line M.04. But, actually, it is not. Condition of Line M.04 could also be implemented as follows:

\[
((\text{ReqS} < \text{Token.C}) \land (\text{Token.N} < \text{Token.C})) \lor ((\text{ReqS} > \text{Token.C}) \land (\text{Token.N} > \text{Token.C}))
\]

This implementation does not require any extra space.

### 4.2 An \(O(\min(n,m))\) Context-Switch Algorithm

Like Algorithm \(mGME\), Algorithm \(nmGME\) in this section is based on Algorithm \(GME\). Processes have an extra variable called \(\text{Counter}\), initialized to zero. Variable \(\text{Counter}\) is reset to zero every time the process executes its exit section. The token has an extra field, \(\text{Token.Cntr}\) which stores the \(\text{Counter}\) value of the (last) process which changed the value of \(\text{Token.N}\).

The idea behind this new algorithm is as follows: Each process \(p\) counts the number of times it tried to access the requested session (Line NM.03). If \(\text{Token.N} = ⊥\) or the request of \(p\) is older than the request carried by the token \((\text{Counter}_p > \text{Token.Cntr};\) see Line MN.04), then \(p\) replaces the (oldest) request in the token by its own request and sets \(\text{Token.Cntr}\) to its own counter value (Line NM.05).

Again, we show that starting from a configuration where a process \(p\) requests for a session, \(p\) can receive the token at most \(2\min(n,m) + 1\) times before \(p\) can enter the critical section. That proves the no lockout property.

**Theorem 6** Algorithm \(mGME\) satisfies the GME specification (as specified in Section 2).

Due to the extra field in the token, the number of bits used by the token is now \(2\lceil \log(m + 1) \rceil + \lceil \log(\min(m,n)) \rceil\) bits. For the same reason, the space com-
Algorithm 4.2 \((\text{nmGME})\) A \(O(\text{min}(n,m))\) GME Algorithm

```plaintext
/**** Lines NM.01 to NM.07 replace Lines 3.14 to 3.17 of Algorithm 3.2 ****/
NM.01 if ReqS \neq \perp then /* A new session is requested */
NM.02 Counter := Counter + 1;
NM.03 if Token.N = \perp \lor Token.Cutr > Counter
NM.04 then Token.N := ReqS; Token.Cutr := Counter
NM.05 endif
NM.06 endif
NM.07 endif
```

The context-switch complexity is \(\text{min}(n,m)\) rounds of passages, and the time complexity is \(2(n - 1)\text{min}(n,m)\) passages. The other complexity results remain the same as for Algorithm \(\text{GME}^\prime\).

### 5 Conclusions

We presented three algorithms to solve the group mutual exclusion problem for anonymous token ring networks. All our algorithms use messages of bounded size. All time related costs of the three proposed algorithms depend on \(n, m,\) and \(\text{min}(n,m)\), respectively. Combined with a protocol which maintains a virtual token circulation ring in a general (rooted) message passing-system (e.g., a depth-first token circulation protocol), these algorithms also solve the group mutual exclusion problem on any arbitrary rooted message-passing system.

**Acknowledgments.** The authors would like to thank the reviewers whose comments greatly helped improve the presentation of the paper.

### References


Sébastien Cantarell is a PhD student in the LRI - CNRS, Université Paris Sud, 91405 Orsay Cedex, France. E-mail: cantarel@lri.fr

Ajoy K. Datta is on the faculty in the Department of Computer Science, University of Nevada Las Vegas, 4505 Maryland Parkway, Box 454019, Las Vegas, Nevada 89154-4019, USA. E-mail: datta@cs.unlv.edu

Franck Petit is on the faculty in the LaRIA, Université de Picardie Jules Verne, 5, rue du Moulin Neuf, F80000 Amiens, France. E-mail: petit@laria.u-picardie.fr

Vincent Villain is on the faculty in the LaRIA, Université de Picardie Jules Verne, 5, rue du Moulin Neuf, F80000 Amiens, France. E-mail: villain@laria.u-picardie.fr
New Bounds on the Size of the Minimum Feedback Vertex Set in Meshes and Butterflies*

IOANNIS CARAGIANNIS
Computer Technology Institute and University of Patras, Greece

CHRISTOS KAKLAMANIS
Computer Technology Institute and University of Patras, Greece

PANAGIOTIS KANELLOPOULOS
Computer Technology Institute and University of Patras, Greece

Abstract

Given a graph $G = (V, E)$, the minimum feedback vertex set $S$ is a subset of vertices of minimum size, whose removal induces an acyclic subgraph $G' = (V\setminus S, E')$. The problem of finding $S$ is NP–complete in general graphs, although polynomial time solutions exist for particular classes of graphs. In this paper we present upper and lower bounds on the size of the minimum feedback vertex set in meshes and butterflies improving results of Luccio [10].

Keywords
Feedback Vertex Set, Mesh, Butterfly

1 Introduction

A feedback vertex set of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ whose removal from $G$ induces an acyclic graph $G' = (V', E')$ with $V' = V\setminus S$ and $E' = \{(u, v) \in E : u, v \in V'\}$. If the cardinality of $S$ is the minimum possible, we call it a minimum feedback vertex set of $G$.

* This work was partially funded by the European Union under IST FET Project ALCOM–FT and Improving RTN Project ARACNE.
The problem has been proved to be NP–complete [4], although polynomial time solutions have been found for particular classes of graphs, e.g., reducible graphs [12], cocomparability and convex bipartite graphs [8], cyclically reducible graphs [15], and interval graphs [9]. For general graphs, the best known approximation algorithm has approximation ratio 2 and is due to Bafna et al. [1].

The problem has important applications to several different contexts. The first application was in combinatorial circuit design (see the discussion in [3]), while it also applies to operating system where resource allocation mechanisms which can prevent deadlocks are desirable. This can be carried out by finding a feedback vertex set in the dependence graph of the tasks that want to use a particular resource and moving the corresponding tasks into a waiting queue so that the new dependence graph becomes acyclic (and deadlock–free) [15]. Clearly, in this situation it is also important to move into the waiting queue the minimum possible number of tasks.

Other applications of the problem are in artificial intelligence (in the constraint satisfaction problem and Bayesian inference [2]), in the study of monopolies in synchronous distributed systems [11], and, recently, in the problem of converters placement in optical networks [5, 14].

1.1 Our results.

In this paper we give new bounds on the size of the minimum vertex set in meshes and butterflies. We present simple alternative ways for computing small feedback vertex sets in 2–dimensional meshes. We also present a method which extends to meshes of higher dimensions. We prove that the bounds we obtain for meshes are asymptotically optimal (provided that the size of the mesh is large) by giving almost matching lower bounds. Our bounds for meshes can also be extended to tori.

For butterflies, we obtain our upper bound by improving the analysis of an algorithm presented in [10]. Using a simple argument, we obtain an almost matching lower bound, thus proving that our upper bound is asymptotically tight (as the dimension of the butterfly increases).

The best previously known bounds for the problem in the particular classes of graphs were due to Luccio [10]. In the following table, we summarize our main results and relate them to previously known ones.

1.2 Outline of the paper.

The rest of the paper is structured as follows. In Section 2 we give a simple and general lower bound argument which yields lower bounds in any graph. Two alternative upper bounds for the 2–dimensional mesh are presented in Section 3. The upper bound for meshes of higher dimensions is presented in Section 4 while the upper bound for the butterfly is given in Section 5.
Table 1: Summary of our results.

<table>
<thead>
<tr>
<th>Network</th>
<th>Previous results (Luccio [10])</th>
<th>Our results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Upper bound</td>
<td>Lower bound</td>
</tr>
<tr>
<td>2-D mesh</td>
<td>$\frac{mn}{2} + \frac{mn}{6} + o(m, n)$</td>
<td>$\frac{mn}{2} - \frac{mn+2}{2}$</td>
</tr>
<tr>
<td>d-D mesh</td>
<td>$1d\frac{2d}{3} + d$</td>
<td>$2d-1</td>
</tr>
<tr>
<td>Butterfly</td>
<td>$11d\frac{2d}{3} + d$</td>
<td>$2d-1</td>
</tr>
</tbody>
</table>

2 Lower bounds

Luccio in [10] proved lower bounds for 2-dimensional meshes and tori, and lower bounds for butterfly networks. Her proofs are based on rather complex arguments.

In this section we present a simple argument useful to prove lower bounds in any network. Using this argument, we extend the known lower bounds for 2–dimensional meshes to meshes of higher dimensions and improve the known lower bound for the size of the minimum feedback vertex set in the butterfly.

Our proofs are based on the following observation.

**Fact 1** A set $S \subseteq V$ is a feedback vertex set of $G$ if and only if the subgraph of $G$ induced by the vertices of $V \setminus S$ is a forest.

**Lemma 1** Any feedback vertex set in a graph $G(V,E)$ with maximum degree $r$ has size at least

$$\frac{|E| - |V| + 1}{r - 1}$$

**Proof.** Let $S \subseteq V$ be a feedback vertex set of $G$. The subgraph $H(V',E')$ of $G$ induced by the vertices in $V' = V \setminus S$ has $|E'| \geq |E| - r|S|$ edges. By Fact 1, $H$ is a forest; thus, we obtain that

$$|V'| \geq |E'| + 1 \Rightarrow$$

$$|V| - |S| \geq |E| - r|S| + 1 \Rightarrow$$

$$|S| \geq \frac{|E| - |V| + 1}{r - 1}$$

We can immediately apply Lemma 1 to meshes, tori, butterflies, and other networks. We only state the lower bounds for the $d$–dimensional mesh and the $d$–dimensional butterfly here.

**Corollary 1** The minimum feedback vertex set of the $d$–dimensional mesh with $n \times n \times \ldots \times n = n^d$ nodes has size at least

$$\frac{(d-1)n^d - dn^{d-1} + 1}{2d-1}$$
Corollary 2 The minimum feedback vertex set of the d–dimensional butterfly has size at least
\[
\frac{(d-1)2^d + 1}{3}
\]
Note that Corollary 2 improves the \(2^{d-1}\left\lfloor \frac{d+1}{2} \right\rfloor\) lower bound presented in [10].

A similar argument was recently used in [3] to prove a lower bound for the hypercube. We believe that using this method, we can achieve lower bounds which are close to the size of the minimum vertex set for graphs in which the average degree is close to the maximum degree. Especially for meshes, we can use some additional arguments to slightly improve the lower bound in Corollary 1 (by an \(o(n^d)\) additive factor). Details are omitted.

3 Upper bounds on the 2–dimensional mesh

For 2–dimensional meshes, Luccio in [10] shows a construction of a feedback vertex set on the \(n \times m\) mesh which has size at most
\[
\frac{mn}{3} + \frac{m+n}{6} + o(m,n).
\]
The construction in [10] computes an optimal feedback vertex set in the case where \(n = m = 2^r - 1\) for some integer \(r > 0\).

Below, we show a way to improve the upper bound in [10]. We begin with a very simple construction.

Consider the partition of the nodes of a two dimensional mesh \(M\) in the three sets \(V_0, V_1, V_2\) defined as
\[
V_k = \{v_{ij} | i + j = k \mod 3\}
\]
for \(k = 0, 1, 2\). We observe that any of the sets \(V_0, V_1,\) and \(V_2\) is a feedback vertex set of \(M = (V(M),E(M))\), since, for any \(k = 0, 1, 2\), the subgraph of \(M\) induced by the vertices in \(V(M) \setminus V_k\) consists of disjoint paths. Since the sets \(V_0, V_1, V_2\) are disjoint, at least one of them has size at most \(\left\lfloor \frac{|V(M)|}{3} \right\rfloor\). We obtain the following result.

Theorem 1 For any integers \(n,m > 0\), the size of the minimum feedback vertex set of a \(n \times n\) mesh is at most \(\left\lfloor \frac{mn}{3} \right\rfloor\).

An example of this construction is depicted in Figure 1. Now, we show a way to improve Theorem 1.

Given a \(n \times m\) mesh \(M\), we apply the above idea to the lower right \((n-2) \times (m-2)\) submesh \(M'\). Let \(S(M')\) be the feedback vertex set of \(M'\), that consists of vertex-disjoint paths. By Theorem 1, we obtain
\[
|S(M')| \leq \frac{(n-2)(m-2)}{3}.
\]
Then, we consider the vertices in the boundary of $M'$, i.e., the $n + m - 3$ vertices with coordinates

$$(1, 0), (1, 1), \ldots, (1, n - 3), (1, n - 2), (2, n - 2), \ldots, (m - 2, n - 2), (m - 1, n - 2)$$

Now we distinguish between three cases according to the feedback vertex set $S(M')$.

Case 1: $(2, n - 3) \in S(M')$. In this case, we define the set of vertices $S'$ to contain the following vertices in the boundary:

$$(1, n - 2), (3, n - 2), \ldots, (n - 1 - (n \mod 2), n - 2)$$

and

$$(1, n - 4), (1, n - 6), \ldots, (1, n \mod 2)$$

i.e., vertex $(1, n - 2)$ and every second vertex on the same row and column.

Case 2: $(2, n - 4) \in S(M')$. In this case, we define the set of vertices $S'$ to contain the following vertices in the boundary:

$$(1, n - 2), (3, n - 2), \ldots, (n - 1 - (n \mod 2), n - 2)$$

and

$$(1, n - 3), (1, n - 5), \ldots, (1, n + 1 \mod 2)$$

i.e., vertex $(1, n - 2)$ and every second vertex on the same row, and vertex $(1, n - 3)$ and every second vertex downwards in the same column.

Case 3: $(3, n - 3) \in S(M')$. This case is symmetric to Case 2.
Figure 2: The feedback vertex set for the $10 \times 10$ mesh.

An example of this construction (for Case 2) is depicted in Figure 2.

It can be easily verified that

$$|S'| \leq \frac{n + m - 1}{2}.$$ 

Now, $S'$ is a feedback vertex set for $M \setminus M'$, since there are no cycles at the top 2 lines (respectively leftmost 2 columns).

The set $S(M') \cup S'$ is a feedback vertex set of $M$ since its removal induces a subgraph of $M$ which has a path consisting of the vertices with coordinates

$$(0, 0), (0, 1), \ldots, (0, n - 1), (1, n - 1), (2, n - 1), \ldots, (m - 1, n - 1)$$

connected to disjoint paths. Clearly, the induced subgraph is a tree.

Furthermore, we have that

$$|S(M') \cup S'| \leq \frac{(n - 2)(m - 2)}{3} + \frac{n + m - 1}{2}$$

$$= \frac{nm}{3} - \frac{n + m - 5}{6}$$

**Theorem 2** For any integer $n, m > 0$, the size of the minimum feedback vertex set of a $m \times n$ mesh is at most

$$\frac{nm}{3} - \frac{n + m - 5}{6}.$$
4 Upper bound on meshes of higher dimension

Our construction of feedback vertex sets of small size in $d$-dimensional meshes $(d \geq 2)$ is based on the existence of distance-3 independent sets of large size.

Given a graph $G = (V,E)$, we call a set of nodes $T \subseteq V$ a distance-3 independent set if the distance between any two nodes in $T$ is at least 3. An example on the 2-dimensional mesh is depicted in Figure 3.

![Figure 3: A distance–3 independent set on the 6 x 7 mesh.](image)

**Lemma 2** In any $d$-dimensional mesh $n \times n \times \cdots \times n$, there exists a distance–3 independent set of size at least $\frac{n^d}{2^d + 1}$.

**Proof.** Consider the partition of the $n^d$ nodes of the mesh into $2^d + 1$ sets of nodes $T_0, T_1, \ldots, T_{2d}$ such that set $T_k$ contains the nodes with coordinates $(x_1, x_2, \ldots, x_d)$ such that

$$\sum_{i=1}^{d} i \cdot x_i = k \pmod{2d + 1}.$$ 

Clearly, the sets $T_0, T_1, \ldots, T_{2d}$ are mutually disjoint. We will show that each one of these sets is a distance–3 independent set of the mesh. In this way, we will obtain the lemma.

Consider two nodes $u$ and $v$ in $T_k$ (for some $k$ such that $0 \leq k \leq 2d$) with coordinates $u = (x_1, x_2, \ldots, x_d)$ and $v = (y_1, y_2, \ldots, y_d)$, respectively. If $u$ and $v$ differ in at least 3 coordinates, then the distance between $u$ and $v$ is at least 3.

Assume that $u$ and $v$ differ in two coordinates, i.e., $x_i \neq y_i$, $x_j \neq y_j$, and $x_m = y_m$ for $m \neq i, j$. Since $u$ and $v$ belong to $T_k$, it holds that

$$\sum_{i=1}^{d} i \cdot x_i = k \pmod{2d + 1}$$
and
\[ \sum_{i=1}^{d} i \cdot y_i = k \pmod{2d + 1}. \]

We obtain that
\[ i \cdot (x_i - y_i) + j \cdot (x_j - y_j) = 0 \pmod{2d + 1}. \]

Now, if \( |x_i - y_i| \geq 2 \) or \( |x_j - y_j| \geq 2 \), the distance between \( u \) and \( v \) is at least 3. Assume that \( |x_i - y_i| = 1 \) and \( |x_j - y_j| = 1 \) which yields \( i \pm j = 0 \pmod{2d + 1} \), a contradiction since \( i \neq j \) and \( 1 \leq i, j \leq d \).

Assume that \( u \) and \( v \) differ in one coordinate, i.e., \( x_i \neq y_i \), and \( x_m = y_m \) for \( m \neq i \). Then,
\[ i \cdot (x_i - y_i) = 0 \pmod{2d + 1}. \]

Now, if \( |x_i - y_i| = 1 \) or \( |x_i - y_i| = 2 \), we obtain that \( i > d \), a contradiction. Thus, the distance between \( u \) and \( v \) is at least 3. This completes the proof of the lemma.

Now, we will show how to construct a feedback vertex set in a \( d \)-dimensional mesh using the construction of a distance–3 independent set presented in Lemma 2.

**Theorem 3** For any integers \( n > 0, d \geq 2 \), the minimum feedback vertex set of a \( d \)-dimensional mesh with \( n \times n \times \ldots \times n = n^d \) nodes has size at most
\[ \frac{d - 1}{2d - 1} \cdot n^d. \]

**Proof.** Given a \( d \)-dimensional mesh \( G_d \) with \( n \times n \times \ldots \times n = n^d \) nodes, we construct a feedback vertex set as follows. We first apply Lemma 2 to obtain a distance–3 independent set \( T \) in the \( (d - 1) \)-dimensional mesh \( G_{d-1} \) of size
\[ |T| \geq \frac{n^{d-1}}{2d - 1}. \]

Now, consider the set \( V' \subseteq V \) of vertices which have the first \( d - 1 \) coordinates identical to the \( d - 1 \) coordinates of some vertex in \( T \). Clearly,
\[ |V'| = n|T|. \]

Consider the two (disjoint) subsets of \( V \setminus V' \)
\[ S_1 = \left\{ v = (x_1, x_2, \ldots, x_d) \in V \setminus V' : \sum_{i=1}^{d} x_i = 0 \pmod{2} \right\}. \]
and
\[ S_2 = \left\{ v = (x_1, x_2, \ldots, x_d) \in V \setminus V' : \sum_{i=1}^{d} x_i = 1 \pmod{2} \right\}. \]

Since \( S_1 \) and \( S_2 \) are disjoint, it holds that
\[ \min\{|S_1|, |S_2|\} \leq \frac{|V \setminus V'|}{2} \tag{3} \]

Using (1), (2), and (3), we obtain that
\[ \min\{|S_1|, |S_2|\} \leq \frac{d-1}{2d-1} n^d. \]

It remains to prove that both \( S_1 \) and \( S_2 \) are feedback vertex sets in \( G_d \). Consider the set \( S_1 \) (the proof for \( S_2 \) is similar). Removing the vertices in \( S_1 \) from \( G_d \) induces a graph \( H \) with a set of vertices \( S_2 \cup V' \). Clearly, \( S_2 \) is an independent set of \( H \), i.e., no two vertices of \( S_2 \) are adjacent. Furthermore, by construction, the vertices in \( V' \) constitute lines; a line for each vertex of \( G_{d-1} \) which belongs to \( T \). Thus, if there exists a cycle in \( H \), then there must be at least a path in \( H \) between vertices belonging to different lines. Note that, by our construction, the distance between two vertices \( v_1, v_2 \) belonging to different lines, is at least 3. Therefore, any path between \( v_1 \) and \( v_2 \) must contain vertices of both \( S_1 \) and \( S_2 \). This contradicts the assumption that \( H \) is the subgraph induced from \( G_d \) by removing the vertices in \( S_1 \). This completes the proof of the theorem.

The above statements can be slightly modified to show that the minimum feedback vertex set in a \( n_1 \times n_2 \times \ldots \times n_d \) mesh has size at most
\[ \frac{d-1}{2d-1} \prod_{i=1}^{d} n_i. \]

Furthermore, our techniques can also be applied to tori by increasing the size of the feedback vertex set by \( O(n^{d-1}) \). Details are omitted.

5 Upper bound on the butterfly

In this section we present an asymptotically optimal upper bound on the size of the minimum feedback vertex set of a \( d \)-dimensional butterfly. Luccio in [10] proved an upper bound of
\[ \left( 2^{d-2} + 2^{d-4} + 2^{d-5} + 1 \right) d = \frac{11d2^d}{32} + d. \]
This bound is \( \Omega(d^2) \) away from the lower bound we proved in Section 2 (see Corollary 2). In what follows we prove that the algorithm presented in [10] actually finds an asymptotically optimal feedback vertex set.

Recall that a \( d \)-dimensional butterfly is a graph \( B_d = (V, E) \) composed by \((d + 1)2^d\) vertices organized in \( d + 1 \) levels of \( 2^d \) vertices each, where \( v_{i,j} \) denotes the \( j \)-th vertex at level \( i \), with \( 0 \leq i \leq d \) and \( 0 \leq j \leq 2^d - 1 \). For \( i > 0 \), \( v_{i,j} \) is connected with the two vertices \( v_{i-1,j} \) and \( v_{i-1,j'} \), where \( j' \) denotes the integer whose binary representation differs from that of \( j \) in only the \( i \)-th bit position.

Algorithm \textsc{Butterfly-FVS}

\[
S := \emptyset \\
\text{for } i := 1 \text{ to } d \text{ do} \\
\quad \text{for } j := 0 \text{ to } 2^d - 1 \text{ do} \\
\quad\quad \text{if } (v_{i-1,j} \notin S) \text{ and } (v_i \notin S) \text{ then } S := S \cup \{v_{i,j}\}
\]

Luccio in [10] proves that after the execution of algorithm \textsc{Butterfly-FVS} on a butterfly \( B_d \), \( S \) is a feedback vertex set of \( B_d \). The execution of the algorithm \textsc{Butterfly–FVS} on a 4–dimensional butterfly is depicted in Figure 4.

Figure 4: The execution of the algorithm on a 4–dimensional butterfly. Black vertices are the vertices of the feedback vertex set \( S \).

We denote by \( b_i(d) \) the number of vertices of level \( i \) which belong in \( S \) and by \( F_d \) the size of \( S \). Clearly,

\[
b_0(d) = 0,
\]

and since there are \( 2^d \) nodes at each level

\[
b_i(d) = \frac{2^d - b_{i-1}(d)}{2}, \quad 1 \leq i \leq d.
\]
Using (4) and the fact that $b_0(d) = 0$ and $b_d(d) \leq 2^{d-1}$, we obtain that

\[
F_d = \sum_{i=0}^{d} b_i(d) = \sum_{i=1}^{d} \frac{2^d - b_{i-1}(d)}{2} = d2^{d-1} - \frac{\sum_{i=0}^{d-1} b_i(d)}{2} = d2^{d-1} - \frac{\sum_{i=0}^{d-1} b_i(d)}{2} + \frac{b_d(d)}{2} \leq d2^{d-1} - \frac{F_d}{2} + 2^{d-2} \Rightarrow
\]

\[
F_d \leq \frac{(d + 1/2)2^d}{3}
\]

Thus, we have proved the following

**Theorem 4** The minimum feedback vertex set of the $d$–dimensional butterfly has size at most

\[
\frac{(d + 1/2)2^d}{3}.
\]

As mentioned in [10], algorithm BUTTERFLY–FVS can also be applied to butterfly–like graphs (i.e., to toroidal butterflies, cube connected cycles). Our analysis improves the known upper bounds on the size of the feedback vertex set of these graphs as well.

**References**


Ioannis Caragiannis is with Computer Technology Institute and the Computer Engineering and Informatics Department, University of Patras, 26500 Rio, Greece. E-mail: caragian@cti.gr

Christos Kaklamanis is with Computer Technology Institute and the Computer Engineering and Informatics Department, University of Patras, 26500 Rio, Greece. E-mail: kakl@cti.gr

Panagiotis Kanellopoulos is with Computer Technology Institute and the Computer Engineering and Informatics Department, University of Patras, 26500 Rio, Greece. E-mail: kanellop@ceid.upatras.gr
Interval Routing in Some Planar Quadrangulations*

VICTOR CHEPOI
Université de la Méditerranée, France

ALEXIS ROLLIN
Université de la Méditerranée, France

Abstract

We show that the subgraphs of the rectilinear grid bounded by simple circuits allow optimal Interval Routing Schemes with at most two circular intervals per edge (2-IRS). We establish that every plane quadrangulations in which all inner vertices have degrees $\geq 4$ has an optimal IRS with at most seven linear intervals per edge (7-LIRS). All routing schemes can be implemented in linear time.

Keywords
interval routing, planar quadrangulations

1 Introduction

A routing scheme is a strategy that assigns to every source-destination pair the path that a message from the source to the destination should take in a given network of $n$ processors. A particular way of implementing routing schemes, called interval routing, has been presented in [10, 7]. In this method, each node is assigned a distinct label from the set $\{1, \ldots, n\}$. Arcs are bidirectional and are labeled with one or several subintervals of the (linear or circular) interval $[1..n]$ so that for any node $v$ the intervals associated with outgoing edges from $v$ are pairwise disjoint and their union covers $[1..n]$. When a message with destination $v$ arrives at node $u \neq v$, it is forwarded on the unique outgoing edge labeled with an interval containing the label of $v$. In most cases, $[1..n]$ is the cyclic interval, i.e., all its subintervals wrap around. Such a scheme is called a circular interval routing scheme (IRS for short). Variants of the scheme include linear interval routing schemes (LIRS), in which $[1..n]$ is viewed as a linear interval; $k$-interval routing

*While preparing this paper, the authors were supported by travel grants from France Telecom R&D and Bayerisch-Franzoesisches Hochschulzentrum.
schemes, in which edges are labeled with at most \( k \) intervals (\( k\)-IRS or \( k\)-LIRS), strict interval routing schemes (\( k\)-SIRS or \( k\)-SLIRS) if any label of an outgoing edge from \( u \) cannot include the label for \( u \). The efficiency of an interval routing scheme is measured in terms of its compactness—the maximum number of intervals constituting the label of an edge. An interval routing scheme for which all messages are routed along shortest paths is called an optimal scheme.

Many regular networks such as complete graphs, grids (alias meshes), hypercubes, complete bipartite graphs, unit interval graphs admit optimal 1-LIRS [1, 3, 7]. Other networks such as trees, rings, grids with one side wrap-around, unit circular-arc graphs, outerplanar graphs, and interval graphs have optimal 1-IRS or 1-SIRS [10, 3, 5, 9]. Finally, in [8] it is shown that the 2-trees allow optimal 3-IRS. For a complete list of related results and concepts see the recent survey [6].

In this note, we design optimal interval routing schemes with small compactness for two classes of planar quadrangulations. First, we consider the rectilinear cells (subgraphs of the rectilinear two-dimensional grid bounded by simple circuits) and show that they allow optimal 2-SIRS. Next we show that every plane graph in which all inner faces are quadrangles and all inner vertices have degrees larger than 3 supports optimal 7-SLIRS. (Missing proofs, figures, and some further results can be found in the full version of the paper.) The quadrangulations \( G \) arising in this paper can be viewed as subgraphs of most popular network topologies (hypercubes, meshes, tori). Even more, each such \( G \) can be represented as a subgraph of respective host graph \( H \) such that the distances in \( G \) and in \( H \) between any pair of vertices of \( G \) coincide (i.e., \( G \) is a distance-preserving subgraph of \( H \)).

Many multi-processor systems (e.g., Intel iPSC860, Intel Paragon) maybe configured as multi-user systems to better utilize the computational power. For this, processors are allocated to users so that no processor is simultaneously used by more than one user. As a result, the respective hypercube or mesh topology is divided into subhypercubes or submeshes specifying a restricted access to a portion of the network for particular users. In general, the set of processors allocated to a specific user may induce an arbitrary subgraph of the network. This implies that messages of a user may pass via processors allocated to another user. In order to avoid this phenomenon, one can force the subgraphs to be connected subhypercubes or submeshes [2, 4]. Since many communication procedures are based on distance criteria, one can further force the respective subgraphs to be distance-preserving. There is a price to pay for this: even if a large number of algorithms for basic communication and routing problems have been developed for all common network architectures, it can be difficult (and different) to solve the corresponding problems when the subgraph allocated to a specific user is arbitrary, connected, or distance-preserving. From this perspective, our note contribute to the routing problem in a multi-user systems with a hypercube, torus or mesh topology in which the subgraph allocated to each user is a quadrangulation defined above.
2 Preliminaries

Let $G = (V, E)$ be a connected undirected graph with $n$ vertices which represents a network. The distance $d_G(u, v)$ between two vertices $u$ and $v$ of $G$ is the length of a shortest path between $u$ and $v$. A (sub)interval $[a, b]$ of the discrete cyclic interval $[1, n]$ is a contiguous subcollection of integers between $a$ and $b$, where $1$ follows $n$ in cyclic order, i.e., $[a, b] := \{i : a \leq i \leq b\}$ if $a \leq b$ and $[a, b] := [1..n] - [b, a]$ otherwise. An $k$-Interval Routing Scheme of $G$ is a pair $R = (L, I)$ verifying the following conditions:

(i) $L$ is a bijection between $V$ and $\{1, \ldots, n\}$, where $L(v)$ is called the label of the vertex $v \in V$;

(ii) $I : E \to 2^{L(V)}$ assigns to each directed edge $(u, v)$ an edge label $I(u, v)$ which is a set containing $k$ or fewer disjoint subintervals of the cyclic (or linear) interval $[1..n]$, such that for each $v \in V$, the intervals associated with the outgoing edges form a partition of $[1..n]$ (possibly excluding $L(v)$);

(iii) for each distinct $u, v \in V$, there exists a path $R(u, v) = (u = x_0, x_1, \ldots, x_t = v)$ such that $L(v) \in I(x_{i-1}, x_i)$ for every $i = 1, \ldots, t$.

For a subset of vertices $X \subseteq V$, set $L(X) := \cup \{L(x) : x \in X\}$.

A plane quadrangulation is a plane graph in which all inner faces are quadrangles. Denote by $Q_4$ the class of plane quadrangulations in which all inner vertices have degree $\geq 4$. A region $R$ of a plane graph is a plane subgraph induced by a finite connected set of inner faces. The boundary $\partial R$ of $R$ is the set of edges which occur in precisely one inner face of $R$. A region $R$ is simply connected if its complement $\mathbb{R}^2 \setminus R$ is connected and if its boundary edges can be ordered to form a single closed path. An important particular case of quadrangulations in $Q_4$ is constituted by the simply connected regions $R$ of the square lattice $\mathbb{Z}^2$, which we call rectilinear cells (for examples see Figure 1).

![Rectilinear Cell](image1.png)  
![Quadrangulation from $Q_4$](image2.png)

Figure 1:

We continue with some additional notions and properties of graphs from $Q_4$. 

---

**Chepoi & Rollin: Interval Routing in Some Planar Quadrangulations**

91
An subgraph $H$ of a graph $G$ is called \textit{gated} if for every vertex $x \notin H$ there exists a vertex $x'$ (the \textit{gate} of $x$) in $H$ such that each vertex $y$ of $H$ is connected with $x$ by a shortest path passing via $x'$. An induced subgraph (or a subset of vertices) $H$ is called \textit{convex} if $H$ includes every shortest path of $G$ between two vertices $u,v$ from $H$. Note that in graphs from $Q_4$ gated and convex sets are the same.

A cut $\{A,B\}$ of $G$ is a bipartition of $V$. If both $A,B$ are convex, we call such a cut \textit{convex} and $A$ and $B$ are called its \textit{halfplanes}. For every convex cut $\{A,B\}$ of $G \in Q_4$ there is a polygonal line $C_i$ along which one can cut the graph $G$ into the subgraphs induced by $A$ and $B$. Denote by $\mathcal{Z} = \{A_1,B_1\}, \{A_2,B_2\}, \ldots, \{A_m,B_m\}$ the collection of convex cuts of $G$ (note that every edge of $G$ is crossed by precisely one convex cut). Let $Z_i$ (resp., $E_i$) be the family of inner faces (resp., edges) of $G$ which are crossed by $C_i$. If we cut the plane along $C_i$, then once entering a face of $Z_i$ via an edge of $E_i$ we must exit this face through a parallel edge and we will never visit this face again. In particular, $C_i$ has no self-intersections.

From more general results follows that $bd(A_i) = A_i \cap Z_i$ and $bd(B_i) = B_i \cap Z_i$ are gated (and therefore convex) paths. We call $bd(A_i)$ and $bd(B_i)$ the \textit{border lines} and $C_i$ the \textit{pseudoline} of the cut $\{A_i,B_i\}$. Every two pseudolines $C_i$ and $C_j$ intersect in at most one point (resp., $Z_i$ and $Z_j$ share at most one common inner face of $G$).

Two cuts $\{A_i,B_i\}$ and $\{A_j,B_j\}$ are \textit{transversal} if all four intersections $A_i \cap A_j, A_i \cap B_j, B_i \cap A_j, B_i \cap B_j$ are non-empty (i.e., if the pseudolines $C_i$ and $C_j$ cross each other), and \textit{laminar} otherwise. A subfamily $\mathcal{C} \subseteq \mathcal{Z}$ of cuts is called \textit{laminar} if every two cuts of $\mathcal{C}$ are laminar (see Figure 2). The importance of laminar collections $\mathcal{C}$ of cuts resides in the underlying tree structure. Namely, cutting $G$ along their pseudolines, we will obtain a partition of $G$ into $k+1$ regions $R_0, R_1, \ldots, R_k$. Define a graph $T(\mathcal{C})$ having $R_0, R_1, \ldots, R_{k+1}$ as a vertex-set and $R_s$ and $R_t$ are adjacent if and only if there is a cut of $\mathcal{C}$ whose two border lines belong one to $R_s$ and another to $R_t$. Since $\mathcal{C}$ is laminar, $T(\mathcal{C})$ is a tree.

![Figure 2: A collection $\mathcal{C}$ of laminar cuts and its tree $T(\mathcal{C})$.](image-url)
3 Routing in rectilinear cells

Throughout this section, \( G = (V, E) \) is a rectilinear cell. The edges of \( G \) are divided into horizontal and vertical. Removing the vertical edges, we obtain a graph consisting of horizontal paths (h-paths) \( hp_0, \ldots, hp_n \). Define the tree \( T^h := T^h(G) \) whose nodes are the h-paths and \( hp_i \) and \( hp_j \) are adjacent in \( T^h \) iff there exists a vertical edge with one end in \( hp_i \) and another in \( hp_j \). If two horizontal paths \( hp_i \) and \( hp_j \) are adjacent in \( T^h \), removing the edges with one end in \( hp_i \) and another in \( hp_j \), we obtain a graph with two connected components \( P_{ij} \) and \( P_{ji} \), where \( hp_i \subseteq P_{ij} \) and \( hp_j \subseteq P_{ji} \). We call \( P_{ij} \) a pocket of \( G \) with respect to \( hp_i \).

The subpath \( s_j \) of \( hp_i \) which is a border line of the cut defined by \( P_{ij} \) and \( P_{ji} \), is called the support of the pocket \( P_{ij} \). Given a vertex \( x \in hp_i \) and a pocket \( P_{ji} \), we say that \( P_{ji} \) is located left (resp., right) from \( x \) if the support of \( P_{ji} \) is left (resp., right) from \( x \). A vertex \( v \) of \( G \) is left from \( x \in hp_i \) if either \( v \) belongs to a pocket left from \( x \) or \( v \in hp_i \) and \( v \) is left from \( x \) (similarly define the vertices right from \( x \)). Let Left\(_i(x)\) and Right\(_i(y)\) denote the set of vertices left from \( x \) and the set of vertices right from \( x \). If \( y \) is the vertex of \( hp_i \) immediately left (resp., right) from \( x \), then Left\(_i(x)\) (resp., Right\(_i(x)\)) consists of the vertices \( z \) such that every shortest path from \( z \) to \( x \) passess via \( y \).

3.1 Vertex labeling

Hereafter \( T^h \) is assumed to be rooted at \( hp_0 \). The labeling algorithm is recursive, starting with the root-path \( hp_0 \). At each stage we traverse some h-path, say \( hp_i \), successively number its vertices, and reserve contiguous intervals of available labels to all pockets of \( hp_i \), except the pocket containing \( hp_0 \). Then we recursively continue the distribution of labels within each pocket of \( hp_i \). Suppose \( P_{ji} \) is a pocket of \( hp_i \) not containing \( hp_0 \). To specify the labeling in \( P_{ji} \), we first label the h-path \( hp_j \) of \( P_{ji} \). At the previous stage, we assigned to \( P_{ji} \) a contiguous interval \([a_j, b_j]\) of numbers. We label the vertices of \( hp_j \) with numbers from \([a_j, b_j]\) and reserve contiguous subintervals of \([a_j, b_j]\) to each pocket of \( hp_j \) not containing \( hp_0 \). The labeling algorithm finishes when all h-paths of \( G \) are labeled. At each stage we have the following picture: a part of vertices of \( G \) is labeled, and the remaining vertices are grouped into pockets with common or distinct supports, and to each such pocket a subinterval of \([1..n]\) is assigned.

Now consider the recursion step in more details. Suppose as above that the current h-path is \( hp_{ij} \), its father in \( T^h \) is the path \( hp_i \), and the circular interval assigned to \( P_{ij} \) is \( I := [a_j, b_j] \). To facilitate the exposition, we assume, without loss of generality, that \( I \) is a linear interval (if not, this can be done by a rotation of the circular interval \([1..n]\)). Notice that \( P_{ij} \) is the pocket of \( h_j \) containing \( hp_0 \). Denote the pockets of \( hp_j \) by \( P_{ij}, \ldots, P_{n_j} \). Every edge and vertex of \( hp_j \) may belong to 0,1, or 2 supports of pockets. We partition the path \( hp_j \) from left to right into maximal by inclusion subpaths \( c_1, c_2, \ldots, c_q \) (called segments) each consisting of
vertices which belong to supports of the same pockets of \( hp_j \). The support of each pocket is a union of one or several consecutive segments.

We start by labeling the rightmost segment \( c_l \) in the support of \( P_{t_j} \) with \( |c_l| \) smallest labels of \([a_j, b_j]\) and update \( I \) by setting \( I := [a_j + |c_l| - 1, b_j] \). To label the remaining segments and pockets of \( hp_j \), we traverse the segments of \( hp_j \) first from \( c_t \) to the left and then from the rightmost segment of \( hp_j \) until \( c_l \). Notice that if \( c_l \) belongs to the support of yet another pocket, the algorithm will end up by labeling this pocket. Let \( c_l \) be the current segment with respect to a chosen direction of labeling. We have three possibilities:

(i) \( c_l \) does not belong to any support of a pocket. Then label \( c_l \) with the interval consisting of first \(|c_l|\) numbers from \( I \) and update \( I \);

(ii) \( c_l \) belongs to the support of exactly one pocket \( P_{t_j} \). If \( P_{t_j} \) was not labeled, then assign to \( P_{t_j} \) an interval consisting of \(|P_{t_j}|\) smallest labels of \( I \), then label \( c_l \) with the next \(|c_l|\) available labels, and update \( I \). Otherwise proceed as in case (i);

(iii) \( c_l \) belongs to the supports of two pockets \( P_{t_{1,j}} \) and \( P_{t_{2,j}} \). If one of these pockets, say \( P_{t_{1,j}} \), has been already labeled, first label \( c_l \) with \(|c_l|\) smallest labels of \( I \), assign to \( P_{t_{2,j}} \) the interval consisting of the next \(|P_{t_{2,j}}|\) consecutive labels of \( I \), and update \( I \). Otherwise, if neither of two pockets is labeled (in this case, the paths \( c_l, s_{t_1}, \) and \( s_{t_2} \) share a common end-vertex), first label the pocket with the largest support, next label the segment \( c_l \), then label the second pocket, and, finally, update \( I \).

After the distribution of labels to the segments and the pockets of \( hp_j \), the vertices in each segment are labeled in a left-to-right order. For a vertex \( v \) of \( G \), let \( L(v) \) be the label of \( v \) given by our algorithm. We can show that for each \( hp_j \), the vertices in one pocket of \( hp_j \) form a single interval in the labeling.

### 3.2 Edge labeling

First, pick a vertical edge \( uv \) of \( G \), with \( u \in hp_i \) and \( v \in hp_j \). Assign to \( I(u, v) \) the interval of labels of \( P_{ji} \), similarly let \( I(v, u) \) be the interval of labels of \( P_{t_j} \). Now, assume that \( uv \) is a horizontal edge of the \( h \)-path \( hp_j \), say \( u \) is left from \( v \). Assign to \( I(u, v) \) the labels of vertices from \( \text{Right}_j(u) \) and to \( I(v, u) \) the labels of vertices from \( \text{Left}_j(v) \). We assert that each \( I(u, v) \) and \( I(v, u) \) occupy one or two circular intervals. If the edge \( uv \) does not lie in the support of some pocket, then \( \text{Left}_j(v) \cup \text{Right}_j(u) = V \) and each \( I(u, v) \) and \( I(v, u) \) is a single circular interval. To consider the remaining cases, suppose that \( u \) and \( v \) are left from \( c_l \). Two cases may occur.

**Case 1.** The edge \( uv \) belongs to the supports of two pockets \( P_{t_{1,j}} \) and \( P_{t_{2,j}} \).

Then \( u \) and \( v \) lie on a common segment \( c_l \), therefore their labels are consecutive numbers. According to the labeling algorithm, one of the pockets, say \( P_{t_{1,j}} \), was labeled before \( c_l \) and the second pocket was labeled immediately after \( c_l \). Hence \( I(v, u) \) occupies two intervals: the vertices of \( c_l \) which are left from \( v \) form one interval and the remaining vertices of \( \text{Left}_j(v) \) forms another interval. The left
endpoint of this interval is the next label after the right endpoint of the label of the pocket $P_{ij}$, and the right endpoint is the largest label of a vertex in a segment or a pocket of $hp_i$ which is left from $c_i$. To show that $I(u,v)$ consists of two intervals, consider the segment $c_k$ which was labeled immediately after the pocket $P_{ij}$. The segment $c_k$ is either the leftmost segment which belongs to the support of $P_{ij}$ or the second leftmost segment (this case occurs when the leftmost segment belongs to the support of a previously labeled pocket). Let $z$ be the rightmost end-vertex of $c_k$. Denote by $P$ the subpath of $hp_i$ comprised between $v$ and $z$. Clearly $P$ belongs to the support of $P_{ij}$. Consider the subset $X$ of $\text{Right}_j(u)$ consisting of $P$ and all pockets having their support in $P$. Due to the labeling algorithm, the labels of vertices from $X$ forms a single contiguous interval $[\mathcal{L}(z), \mathcal{L}(v)]$. Indeed, after labeling $P_{ij}$, the algorithm labels $c_k$ first, next it labels the pocket having the segment $c_k$ as a support (if such a pocket exists), and then it consider the segment immediately left from $c_k$. It treats this segment in the same way as $c_k$. The labeling of $X$ will finish when the algorithm arrives at the segment $c_i$. Thus the label of $X$ is indeed $[\mathcal{L}(z), \mathcal{L}(v)]$. Using similar arguments, one can show that $\text{Right}_j(u) - X = \text{Right}_j(z)$ constitutes a single circular interval.

Case 2. The edge $uv$ belongs to the support of a single pocket $P_{ij}$.

The analysis of this case is similar to Case 1. Concluding, we have established that for every edge $uv$ of $G$, the labels $I(u,v)$ and $I(v,u)$ occupy at most two contiguous circular intervals each.

### 3.3 Routing

We show that for two vertices $u, v$ of $G$ there is a shortest path $R(u,v) = (u = x_0, x_1, \ldots, x_k = v)$ such that $\mathcal{L}(v) \subseteq I(x_{i-1}, x_i) \ (i = 1, \ldots, k)$. For this, proceed by induction on $k := d_G(u,v)$. If $u$ and $v$ are adjacent, this is trivial. Now, let $k \geq 2$. Consider the $h$-path $hp_i$ passing via $u$. Let $w$ be the vertex of $hp_i$ immediately right from $u$. As we noticed already (and this easily follows from the fact that $h$-paths are gaded), $w$ lies on a shortest path between $u$ and any vertex located right from $u$. Therefore, if $v \in \text{Right}_i(u)$, then $\mathcal{L}(v)$ belongs to the label of the directed edge $(u,w)$. Since $d_G(u,v) = d_G(w,v) + 1$, applying the induction assumption to the couple $w,v$, we obtain the shortest path $R(u,v) = \{u\} \cup R(w,v)$ along which messages from the source $u$ are routed to the destination $v$. Now, suppose that $v$ belongs to a pocket $P_{ji}$ whose support $s_j$ contains the vertex $u$. Let $z$ be the neighbour of $u$ in $P_{ji}$. According to the algorithm, the label of the directed edge $(u,z)$ is precisely the label of the pocket $P_{ji}$. In particular, $\mathcal{L}(v) \subseteq I(u,z)$. Since $z$ is the gate of $u$ in $P_{ji}$, we have $d_G(u,v) = d_G(z,v) + 1$. Again, applying the induction hypothesis to the pair $z,v$ we obtain the shortest path $R(z,v)$. Adjoin to this path the edge $uz$ to derive the desired shortest path $R(u,v)$. Notice that the shortest paths used in the routing “prefer” vertical edges provided the current vertex and the destination $v$ are in a common pocket and use horizontal edges only to move towards the current pocket hosting $v$. Summarizing, here is the main result.
of this section:

**Theorem 1** For rectilinear cells, the described routing scheme $R = (L, I)$ is an optimal 2-SIRS (3-SLIRS) and can be constructed in linear time.

The results of this section can be extended in a straightforward way to biconnected quadrangulations $G \in Q_4$ in which all inner vertices have degree 4 and all boundary vertices have degree $\leq 4$.

**Open Question.** We conjecture that there exists a rectilinear cell not admitting an optimal 1-SIRS.

## 4 Routing in quadrangulations from $Q_4$

In this section we present an optimal 7-SLIRS for graphs $G = (V, E)$ from $Q_4$. For this, we construct a collection $C \subseteq \mathcal{Z}$ of laminar cuts which partitions $G$ into rectilinear or pseudorectilinear cells. Each pseudorectilinear cell is further subdivided into rectilinear cells using a new family of laminar cuts (but with respect to this cell only). Applying the algorithm from Section 3, the fact that the regions in the resulting partition are gated, and the treelike structure of the family $C$, we can establish the desired routing scheme.

### 4.1 Construction of $C$

The collection $C$ is constructed step by step, by adding each time a new convex cut which is laminar to previously defined cuts. We take an arbitrary convex cut as the first cut of $C$ and place its halves into a queue $Q$. Suppose we have defined a collection $C$ of $i$ laminar cuts. Their pseudolines partition $G$ into $i + 1$ regions. Several of these regions are in the current queue. Pick the region $R$ at the front of $Q$ and search it for a new laminar convex cut. If such a cut is not found, we delete $R$ from $Q$. Now suppose a new laminar cut $\{A, B\}$ has been found. Its pseudoline $C$ partitions $R$ into two regions $R'$ and $R''$. One of these regions, say $R''$, is a halfplane of $G$ (hence it can be further treated as the halves of the first cut). We add $\{A, B\}$ to $C$, replace $R$ by $R'$ at the front of $Q$ and add the region $R''$ at the back of the queue. The algorithm stops when $Q$ is empty.

Notice that a region $R_0 := R$ which just arrived at the front of $Q$ is always a halfplane. Let $R^+$ be the first region removed from the front of the queue after the arrival of $R$ in head. Obviously $R^+ \subseteq R$ is a region in the final subdivision of $G$. We call the intermediate steps between handling $R$ and removing $R^+$ from $Q$ a *phase*. During a phase we cut off disjoint halfplanes from $R$ and add their defining cuts to $C$. The updated region will be also denoted by $R$. To prove the correctness of the algorithm, it suffices to precise the evolution of the region $R$ in front of $Q$ during the phase and to establish the structure of the final region $R^+$.

Denote by $L_0$ the border line of the cut of $G$ whose halfplane is the initial region $R_0$. The complement of $L_0$ in the boundary of the current region $R$ is called
the beach line of $R$ and is denoted by $\beta(R)$. This path is an alternating sequence of subpaths of the boundary of $G$ and subpaths of border lines of laminar cuts found within the phase but before handling the current region $R$. Finally, the sweep front $\alpha(R)$ consists of one or several paths (called sweep lines) with both end-vertices on the beach line. Together with $L_0$ and some subpaths in the beach line, the sweep front bounds a subregion $R^+$ of $R$. The vertices in $R^+$ are precisely those vertices of $R$ which have been already swept. For each vertex $v \in \alpha(R)$, we denote by $\deg^+(v)$ the number of external neighbours of $v$, i.e., neighbours located in $R$ but outside $R^+ \cup \alpha(R)$. In one iteration of the phase, the vertices of current sweep line $L$ are considered one after other. If a current vertex $v \in L$ has $\deg^+(v) \geq 2$, and there exists a cut passing in the neighbourhood of $v$ such that one of its halfplanes is disjoint from $R^+$, then we add this cut to $C$, update the region $R$, its beach line, and $R^+$. Finally, replace in the sweep front the path containing $v$ by two its subpaths, and start a new iteration of the same phase. Otherwise, if all vertices in a path of the sweep front have been considered without finding a new cut, we advance this path, add the swept line to $R^+$ update the sweep front, and start a new iteration. In Figure 3 we present an example of a partially swept region $R$.

Each path $L$ of $\alpha(R)$ contains both its end-vertices on the beach line. These vertices divide the boundary of $R$ into two chains $C_L, C'_L$. One of these chains, say $C_L$ is disjoint from $L_0$. In analogy with rectilinear cells, call the region $R_L$ bounded by $L$ and $C_L$ a pocket of $L$. Clearly $R_L \cap R^+ = L$. At each iteration of the phase, we take care to preserve the following structural invariant: for each path $L$ of the sweep front $\alpha(R)$, the chain $C_L$ either consists of a single subpath of $\partial G$, or of a subpath of $\partial G$ and a subpath $P$ of a border line of some cut of $C$, or of two subpaths $P', P''$ of border lines of two cuts of $C$ and a subpath of $\partial G$ in between. In first case, the pocket $R_L$ is a halfplane. In second and third cases, we call $R_L$ a bigon and a trigon, respectively.
At the beginning of the phase, $R$ is a halfplane. The sweep front $\alpha(R)$ consists solely of $L_0$. The region $R^+$ is empty and the pocket of $L_0$ is the whole region $R$. Now, let a current region $R$ be given. Suppose its beach line $\beta(R)$ and the sweep front $\alpha(R)$ are defined, and a current line $L \in \alpha(R)$ has to be considered ($L = L_0$ at the beginning). Denote by $v_0, v_0'$ the end-vertices of $L$. We traverse the vertices of the line $L$ from one end to another (to fix an orientation, say from left to right), and stop at the first vertex (if it exists) with at least two external neighbours. In dependence of its existence, location, and the form of the pocket $R_L$, several cases may occur. We distinguish three main cases: $R_L$ is a halfplane, a bigon, or a trigon. Due to the space limitations, only one subcase of the case of trigons is considered in some details (for missing figures and cases see the full version of the paper).

Let $R$ be a trigon bounded by a line $L$ of the sweep front, two subpaths $P^I, P^II$ in the beach line (each of them is a subpath of a border line of a cut from $C$) and a subpath of $\partial G$. Suppose $P^I$ bounds $R$ from left and $P^II$ bounds $R$ from right. Let $P^I \cap L = \{v'_0\}$ and $P^II \cap L = \{v''_0\}$. Denote by $u'_0$ and $u''_0$ the neighbours of $v'_0$ and $v''_0$ in $P^I$ and $P^II$, respectively. In this case, the path $L$ either belongs to the border line of a single cut or $L = L_1 \cup L_2$, where the paths $L_1$ and $L_2$ belong to border lines of two transversal cuts (then let $v_0$ be the common vertex of $L_1$ and $L_2$). In the first case we say that $L$ is an 1-line, in the second case $L$ is called a 2-line. While sweeping a trigon, the evolution of the sweep line is roughly the following: at the beginning it is an 1-line and it preserve this form (or it can be replaced by several 1-lines) until we will come to an 1-line which contains exactly one vertex with at least two external neighbours and this vertex is an inner vertex of degree 5. Then the new sweep line becomes a 2-line. It can remain a 2-line during several iterations, until on this line we will find a vertex of degree at least 5. Then it is replaced by two 1-lines or by an 1-line and a 2-line. The region $R^+$ obtained at the end of the phase is not always a rectilinear cell, it may contain a certain number of inner vertices of degree 5. These vertices are kept in the list $D(R^+)$. 

{f Case Trigon 3}: The unique external neighbour of $v'_0$ is the vertex $u'_0 \in P$. Let $v$ be the leftmost vertex of $L$ having degree larger or equal than 5.

{f Subcase (i):} $v$ has degree larger or equal than 6 (see Figure 4).

Let $u, w$ be the two leftmost external neighbours of $v$. Add to $C$ the cut $\{A_j, B_j\}$, which is defined either by the edge $vw$ if $L$ is an 1-path or $L$ is a 2-path and $v \in L_1$, or by the edge $vu$ if $v \in L_2$. In both cases, let $v \in A_j$. Update $R$ by letting $R := R \setminus B_j$ and add $bd(A_j)$ to the beach line. In the first two cases, replace in $\alpha(R)$ the path $L$ by two paths $L'$ and $L''$, where $L'$ is the subpath of $bd(B_j)$ comprised between $u'_0$ and $u$, and $L''$ is the subpath of $L$ comprised between $v$ and $v''_0$. The pockets of $L'$ and $L''$ are trigons. Notice that in the first case both $L'$ and $L''$ are 1-lines and in the second case $L'$ is an 1-line and $L''$ is a 2-line. In both cases, add the subpath of $L$ between $u'_0$ and $u$ to $R^+$. In the third case, replace $L$ by a path $L'$ induced by the external neighbours of all vertices of $L_1$ and a subpath $L''$ of $L_2$ comprised between $v$ and $v''_0$. Add $L_1$ and the subpath of $L_2$ between $v_0$ and $v$ to the region $R^+$. In this case, $L'$ and $L''$ are 1-paths and their pockets are trigons.
Subcase (ii): \( v \) has degree 5.

Let \( u, w \) be the external neighbours of \( v \) and let \( vw \in E_j \). If \( L \) is a 2-path, then proceed as in Subcase (i) with the unique difference that if \( v \in L_{ii} \), then \( L' \) is the subpath of \( L \) comprised between \( v_0 \) and the rightmost common vertex \( z \) of \( L \) and \( bd(A_j) \). In this case, we additionally add the subpath between \( v \) and \( x \) to \( R \). So, further assume that \( L \) is an 1-path. We sweep \( L \) from \( v \) to right to find the next vertex \( x \in L \) (if it exists) with zero or at least two external neighbours. If \( x \) does not have external neighbours, then we just follow the case Trigon 2. Now, suppose the vertex \( x \) has at least two external neighbours. Denote by \( y, z \) the first and the second leftmost external neighbours of \( x \). Obviously the edge \( xy \) belongs to the equivalence class \( E_j \). Again proceed as before: add the \( j \)th cut to \( C \) and update correspondingly the beach line, the sweeping front and the region \( R \).

Finally suppose that every vertex of \( L \setminus \{ v \} \) has exactly one external neighbour. Delete the path \( L \) from the sweep front and add it to the region \( R \). Add to the sweep front the 2-path \( L' \) induced by the external neighbours of the vertices from \( L \). The pocket of \( L' \) is also a trigon. Add \( v \) to \( D(R) \).

**Lemma 1** \( C \) is a collection of laminar cuts.

### 4.2 Dealing with pseudorectilinear cells

Assume that the final collection \( C \) consists of \( k \) laminar cuts \( \{A_{i_1}, B_{i_1}\}, \ldots, \{A_{i_k}, B_{i_k}\} \), whose pseudolines cut \( G \) into \( k + 1 \) regions (called cells) \( R_0^+, R_1^+, \ldots, R_k^+ \) numbered in order of their creation. Since every \( R_j^+ \) is the
intersection of halfplanes containing it, \( R^+_j \) is a gated subgraph of \( G \).

As we noticed, the biconnected components of the regions \( R^+_0, R^+_1, \ldots, R^+_k \) may contain a certain number of inner or boundary vertices of degree 5. These vertices are kept in the lists \( D(R^+_j), j = 0, \ldots, k \). To partition every biconnected component of a pseudorectilinear cell \( R^+_j \) into rectilinear cells, we construct a collection \( C(R^+_j) \) of laminar convex cuts of \( R^+_j \) such that for each vertex \( v \) of degree 5 there is a cut from \( C(R^+_j) \) which includes an edge incident to \( v \). Notice that every cut of \( C(R^+_j) \) extends in a unique way to a cut from \( Z \).

From the sweeping algorithm we know that when the inner vertex \( v \) is added to the list \( D(R^+_j) \), the sweep line \( L \) passing via \( v \) does not contain other vertices of degree \( \geq 5 \) and that \( L \) is an 1-line. Denote by \( L' \) the sweep line considered before \( L \) and such that every vertex of \( L \) has a unique neighbour in \( L' \). Let \( v' \) be the neighbour of \( v \) in \( L' \) and suppose that \( v' \notin E_i \). Add the cut \( \{ A_i \cap R^+_j, B_i \cap R^+_j \} \) of \( R^+_j \) to \( C(R^+_j) \). Set \( A_{jl} := A_i \cap R^+_j, B_{jl} := B_i \cap R^+_j \). Now, suppose that \( v \in D(R^+_j) \) lies on the boundary of \( R^+_j \). In this case, we consider one or two edges incident to \( v \) whose other end-vertices lie on forthcoming sweep lines. Then add the cut(s) defined by this edge (or these edges) to \( C(R^+_j) \). Pick another vertex \( w \) of degree 5 and let \( \{ A_{jl}, B_{jl} \} \) be the cut defined by the edge \( w \in E_i \) where \( w' \) is defined in the same way as \( v' \). These two cuts are necessarily laminar in \( R^+_j \), otherwise we obtain a subregion of \( R^+_j \) with at most three corners. Thus \( C(R^+_j) = \{ \{ A_{jl}, B_{jl} \}, \ldots, \{ A_{jk}, B_{jk} \} \} \) is indeed a collection of laminar cuts whose pseudolines partition the cell \( R^+_j \) into the rectilinear cells \( SR^+_{jl}, \ldots, SR^+_{jk} \). Denote by \( T(C(R^+_j)) \) the tree whose nodes are these rectilinear cells and two nodes are adjacent iff each of the corresponding cells contains a border line of a cut from \( C(R^+_j) \).

### 4.3 Vertex labeling

First we reserve contiguous intervals of labels to each of the pseudorectilinear cells \( R^+_0, R^+_1, \ldots, R^+_k \). Due to the implementation of the queue \( Q \), the numbering of these regions corresponds to a BFS traversal of the tree \( T(C) \) rooted at \( R^+_0 \). According to the algorithm from Subsection 4.1, \( R^+_0 \) is obtained from a half-plane of \( G \), say \( A_{i_0} \), by cutting off from \( A_{i_0} \) some disjoint halfplanes \( B_{i_1}, \ldots, B_{i_n} \). Each of these halfplanes comprises the cells contained in a subtree of \( T(C) \) obtained by removing the root \( R^+_0 \). To \( R^+_0 \) we reserve the labels from the interval \([1..|R^+_0|] \). Now, we traverse the boundary of \( R^+_0 \) in counterclockwise order starting from \( B_{i_1} \) and reserve contiguous cyclic subintervals of the interval \([|R^+_0| + 1..n]\) to each of the halfplanes \( B_{i_j}, j = 1, \ldots, s \). Assume without loss of generality that \( B_{i_1}, B_{i_2}, \ldots, B_{i_s} \) is the order in which we meet the border lines of the complements of these halfplanes. To \( B_{i_1} \) we assign the interval \([|R^+_0| + 1..|B_{i_1}| + |B_{i_2}|] \) and to \( B_{i_j} (j = 2, \ldots, s) \) we assign the interval \([|B_{i_1}| + \sum_{i=1}^{j-1} |B_{i_i}| + 1..|R^+_0| + \sum_{i=1}^{j-1} |B_{i_i}| \].
The cyclic interval assigned to each of the halfplanes $B_{ij}$, $j = 1, \ldots, s$, is further subdivided into smaller contiguous intervals using the same procedure: the interval with smallest available labels is reserved to the cell $R^+_{j\ell}$ obtained at the end of the phase on which the halfplane $B_{ij}$ is swept, while the remaining labels are distributed in intervals among the halfplanes which are cut off from $B_{ij}$ during that phase (the order of affecting intervals is the same as for the root cell, i.e., by traversing in counterclockwise way the beach line of $R^+_{j\ell}$). Notice the following elementary but basic property of the resulting assignment.

**Lemma 2** Let $x, y$ be two arbitrary vertices of $\partial R^+_{j\ell}$. Let $P$ be one of the subpaths of $\partial R^+_{j\ell}$ comprised between $x$ and $y$, and consider all halfplanes disjoint from $\partial R^+_{j\ell}$ such that the border lines of their complements are completely contained in $P$. Then the union of labels of these halfplanes occupies one or two linear subintervals of $[1..n]$.

At the end of this first part of the labeling algorithm, to each cell $R^+_{j\ell}$ ($j = 0, \ldots, k$) is assigned one circular interval $L(R^+_{j\ell})$, such that $L(R^+_{0\ell}), \ldots, L(R^+_{k\ell})$ form a partition of $[1..n]$. Moreover, the unions of labels of cells in each of two subtrees obtained by removing an arbitrary edge of $T(C)$ constitute two complementary subintervals of the cyclic interval $[1..n]$. If $R^+_{j\ell}$ is a rectilinear cell (i.e., every inner vertex has degree 4 and every boundary vertex has degree at most 4), then applying the algorithm from Subsection 3.1, we label the vertices $v$ of $R^+_{j\ell}$ with numbers $L(v)$ from the interval $L(R^+_{j\ell})$.

Now, let $R^+_{j\ell}$ be a pseudorectilinear cell. In order to label its vertices, we previously assign disjoint subintervals of $L(R^+_{j\ell})$ to the rectilinear cells $SR^+_{j0\ell}, \ldots, SR^+_{jk\ell}$ into which $R^+_{j\ell}$ is partitioned by the pseudolines of the cuts from $C(R^+_{j\ell})$. This can be done using a similar procedure: we reserve a sufficient amount of smallest available labels to the first subcell $SR^+_{j0\ell}$, and the rest of labels is distributed in intervals among the halfplanes of cuts from $C(R^+_{j\ell})$ which are disjoint from $SR^+_{j0\ell}$ and such that the border lines of their complements are contained in $\partial(SR^+_{j0\ell})$. Again the order of labeling of these halfplanes follows the counterclockwise traversal of $\partial(SR^+_{j0\ell})$. These intervals are further subdivided provided new regions bounded by cuts from $C(R^+_{j\ell})$ are found. As a result, we assign to each $SR^+_{j\ell}$ ($l = 0, \ldots, k_j$) a single subinterval $L(SR^+_{j\ell})$, which together partition the interval $L(R^+_{j\ell})$. Since each $SR^+_{j\ell}$ is a rectilinear cell, the final labeling of the vertices from $SR^+_{j\ell}$ is done by the algorithm from Subsection 3.1.

One can note that the labeling of the vertices of a quadrangulation $G \in Q_k$ is performed via a three-level treelike structure. The tree $T(C)$ whose nodes are the cells $R^+_{0\ell}, \ldots, R^+_{k\ell}$ is the main tree. For each node $R^+_{j\ell}$ we construct the second-level tree $T(C(R^+_{j\ell}))$ whose nodes are the rectilinear cells $SR^+_{j\ell}, l = 0, \ldots, k_j$. The vertices of each $SR^+_{j\ell}$ are further grouped into horizontal paths, and the incidence relation between these paths define the tree $T^h(SR^+_{j\ell})$, the third-level tree.
4.4  Edge labeling and routing

Pick an arbitrary vertex $u$ of $G$, say $u \in R_j^+$. Assume aditionally that $u$ belongs to the rectilinear cell $SR_{jl}^+$. We show how to label the outgoing edges $(u,v)$. For this, divide the label $I(u,v)$ into three groups $I(u,v) := I_1(u,v) \cup I_2(u,v) \cup I_3(u,v)$ (one or two groups may be empty). If both $u$ and $v$ belong to a common rectilinear cell $SR_{jl}^+$, then $I_1(u,v)$ consists of the labels of vertices $x \in SR_{jl}^+$ such that the messages from $u$ to $x$ will be routed via $v$, otherwise $I_1(u,v)$ is empty. If $v \in R_j^+$, then $I_2(u,v)$ is the union of labels of all vertices $x \in R_j^+ \setminus SR_{jl}^+$ such that the messages from $u$ to $x$ will be routed via $v$ ($I_2(u,v)$ is empty if $v \notin R_j^+$). Finally, $I_3(u,v)$ consists of the labels of all vertices $x \in G \setminus R_j^+$ such that the messages from $u$ to $x$ are routed via $v$.

The label $I_1(u,v)$ is computed using the algorithm from Section 3 which provides a labeling of all vertices and oriented edges of the rectilinear cell $SR_{jl}^+$. Hence each $I_1(u,v)$ occupies one or two cyclic subintervals of the interval $L(SR_{jl}^+)$. In the global routing scheme we are allowed to use only subintervals of $[1..n]$, thus we must transform these cyclic subintervals of $L(SR_{jl}^+)$ into at most three linear subintervals of $[1..n]$. Since $SR_{jl}^+$ is convex and routing in each $SR_{jl}^+$ is done as in the rectilinear cells, we immediately deduce that the labels from the first group $I_1$ assure the optimal routing inside each rectilinear cell $R_j^+$.

Now assume that $uv \in E_i$ for a cut $\{A_{ij}, B_{ij}\} \in C$. Suppose without loss of generality that $u \in R_{ij}^+ \subseteq A_{ij}$ and $v \in B_{ij}$. Set $I_3(u,v) = L(B_{ij})$ and $I_3(v,u) = L(A_{ij})$. From Subsection 4.4. we know that the labels assigned to $B_{ij}$ and $A_{ij}$ constitute two complementary circular subintervals of $[1..n]$. Since $v$ is closer than $u$ from every vertex $x \in B_{ij}$, we obtain the required property of routing.

It remains to deal with outgoing edges $(u,v)$ which belong to the pseudo-rectilinear cell $R_j^+$, more precisely, to one biconnected component of this cell (one can assume without loss of generality that $R_j^+$ itself is biconnected). Assume that $R_j^+$ was constructed in the phase during which the halfplane $A_{ij}$ was swept. Suppose that at this phase the cuts $C_j = \{\{A_{ij}, B_{ij}\}, \{A_{ij+1}, B_{ij+1}\}, \ldots, \{A_{ij+s}, B_{ij+s}\}\}$ have been added to $C$, where $B_{ij}, B_{ij+1}, \ldots, B_{ij+s}$ are disjoint from $R_j^+$. This means that $R_j^+ = A_{ij} \setminus (B_{ij} \cup B_{ij+1} \cup \ldots \cup B_{ij+s})$. The numbering of the halfplanes $B_{ij}, B_{ij+1}, B_{ij+2}, \ldots, B_{ij+s}$ is the same as in Subsection 4.3., i.e., following the counterclockwise traversal of $\partial R_j^+$. From the sweeping algorithm of Subsection 4.1. we know that $\{A_{ij}, B_{ij}\} \in C$, hence $L(A_{ij})$ and $L(B_{ij})$ form two complementary subintervals of the cyclic interval $[1..n]$.

The vertex $u$ has at most five neighbours in $R_j^+$, say $v_1, v_2, v_3, v_4, v_5$. Consider the classes $E_p$ defined by the edges $uv_p$, $p \leq 5$. Since $R_j^+$ is convex, every $E_p$ shares precisely two edges with $\partial R_j^+$, the leftmost edge $le_p$ and the rightmost edge $re_p$. These are either two edges of $\partial G$, or one edge of $\partial G$ and one edge of a border line of a cut from $C_j$, or two edges from the border lines of two distinct
cuts of $C_i$. Hence $\{A_i, B_p\}$ is laminar with all cuts of $C_i$ except one or two cuts. If $u$ is an inner vertex of degree 4, the four cuts defined by the edges incident to $u$ divide $R^+_i$ into eight bigons. Similarly, if $u$ has degree 5, we will obtain ten bigons (the case $u \in \partial R^+_i$ is similar, even easier). Consequently, the boundary of the region $R^+_i$ will be partitioned into eight or ten paths. Since the cuts defined by two edges $uv$ and $uq$ lying in a common 4-face are transversal and $Z$ does not contain three pairwise transversal cuts, $E_p$ and $E_q$ cannot share common edges with the border line of the same halfplane. To $I_3(u, v_1)$ we assign the labels of all halfplanes $B_{il+q}$ such that $bd(A_{il+q})$ is contained entirely in the subpath of $\partial R^+_i$ comprised between the edges $le_1$ and $re_1$. If $le_1 \in \partial G_e$ to $I_3(u, v_2)$ we assign the labels of all halfplanes $B_{il+p}$ such that $bd(A_{il+p})$ is contained entirely in the subpath of $\partial R^+_i$ comprised between $le_1$ and $re_2$. Otherwise, if $le_1 \in bd(A_{il+q})$, then assign to $I_3(u, v_2)$ the label of the halfplane $B_{il+p}$ plus the labels of all halfplanes $B_{il+p}$ such that $bd(A_{il+p})$ is contained entirely in the subpath of $\partial R^+_i$ comprised between $le_1$ and $re_2$. In a similar way, distribute labels to the rest of outgoing arcs. From Lemma 2, we conclude that each $I_3(u, v_p)$ occupies one or two linear subintervals of $[1..n]$. Notice also that if $L(x) \in I_3(u, v_p)$, then $v_p$ is closer to $x$ than $u$, because $x \in V(v_p, u) = B_p$.

In a similar way, we specify the group $I_2$. Suppose that $R^+_i$ was partitioned into the rectilinear cells $SR^i_{j_1}, \ldots, SR^i_{j_k}$ using the collection of laminar cuts $C(R^+_i)$. Let $\{A_{j_1}, B_{j_1}, \ldots, A_{j_k}, B_{j_k}\}$ be the cuts from $C(R^+_i)$ such that $bd(A_{j_1}) \cup \ldots \cup bd(A_{j_k}) \subset \partial(SR^i_{j_k})$. If $uv$ participates in one of these cuts, then set $I_2(u, v) := L(B_{j_k})$ and $I_2(v, u) := L(A_{j_k})$. Each of these labels occupies a single cyclic subinterval of $L(R^+_i)$, therefore one or two linear subintervals of $[1..n]$. It remains to define $I_2(u, v_p)$ for neighbours $v_p$ of $u$ inside the rectilinear cell $SR^i_{j_k}$. If $u$ is an inner vertex of $SR^i_{j_k}$ then it has four neighbours $v_1, v_2, v_3, v_4$. The case when $u \in \partial(SR^i_{j_k})$ is completely similar. Each $I_2(u, v_p)$ occupies a single circular subinterval of $L(R^+_i)$, therefore two linear subintervals of $[1..n]$. Since $R^+_i$ is convex, this shows that routing messages between two vertices in different rectilinear subcells of $R^+_i$ can be done using at most two intervals per edge (from the group $I_2$).

Note that in order to route the message from $u$ to the destination outside the pseudorectilinear cell $R^+_i$ or outside the rectilinear cell $SR^i_{j_k}$, we send the message to a neighbour of $u$ which has the same gate as $u$ in the halfplane of $G$ or of $R^+_i$. Consequently, the message from $u$ will be sent to its gate in the respective halfplane. This establishes the optimality of the routing scheme. Summarizing, here is the main result of this note.

**Theorem 2** For a quadrangulation $G \in Q_4$, the described routing scheme $R_c = (L, I)$ is an optimal 7-SLIRS.
References


Victor Chepoi is with the Laboratoire d’Informatique de Marseille, Université de la Méditerranée, Faculté des Sciences de Luminy, F-13288 Marseille Cedex 9, France. E-mail: chepoi@lim.univ-mrs.fr

Alexis Rollin is with the Laboratoire d’Informatique de Marseille, Université de la Méditerranée, Faculté des Sciences de Luminy, F-13288 Marseille Cedex 9, France. E-mail: rollin@lim.univ-mrs.fr
Worst-Case Bounds for Blind Broadcasting in Small-Degree Networks

PETER DAMASCHKE
Chalmers University, Göteborg, Sweden

Abstract

Blind broadcasting is the task of delivering a message from a source to all the \( n \) nodes in a network, where nodes are not aware of the network topology. We derive upper and lower bounds for the worst-case broadcast time, in terms of size \( n \) and depth \( d \) of the network (i.e. eccentricity of the source). Provided that the degree of nodes is bounded by a small constant, these results improve upon the general upper bound of \( 2n - d \). For degree up to 5 we get tight bounds.

Keywords

broadcasting, one-port model, unknown topology, small degree, full synchronization

1 Blind Broadcasting

Broadcasting is a fundamental topic in network communication. A network can be modeled by an undirected graph whose nodes and edges are sites and communication links, respectively. One distinguished node called the source has to communicate some message to all nodes in the network. A node is called informed if it has received the message. Let \( n \) be the size (number of nodes) of the network, and \( d \) the depth (also called eccentricity), i.e. the maximum distance of nodes from the source.

**Model:** We adopt the following one-port model: Communication is performed in synchronized rounds. In every round, each node can send a message to one of its neighbors, that is, only one message at a time. However two messages may pass an edge simultaneously in both directions.

We assume that the nodes have no knowledge about the network topology. (This is an appropriate assumption in large networks which are permanently changing.) Each node only is only aware of its own degree and can identify its incident edges, i.e. assign numbers to them.

In *blind broadcasting* all nodes have to be informed, but no acknowledgement of receipt is required. It is well known and easy to realize that blind broadcasting...
is finished if and only if the message has passed through every edge in at least one
direction [5, 8]. An edge that has already carried the message is called saturated, and a node is called saturated if all incident edges are saturated.

Since, in our model, all edges incident to the same node “look equal”, there is a
unique protocol for blind broadcasting, called BB: In every round, each informed
but unsaturated node sends the message through one of the unsaturated incident
edges. The number of rounds is solely determined by an adversary who chooses,
for every informed node, the unsaturated edge that shall be used next by this node.

It has been shown in [8] that BB requires at most $2n - d$ rounds, and that fac-
tor 2 is asymptotically optimal in the worst case. Open questions in [8] addressed
the influence of other network parameters and of partial knowledge of the net-
work to the broadcast time. Results for the case that nodes know their respective
neighborhoods of bounded radius are obtained in [5]. In the present paper we con-
sider networks where all node degrees are bounded by some fixed small $\Delta$, as it is
typical in many real networks. Other related work deals e.g. with broadcasting in
networks with unlabeled nodes and edges, but with known topology [2, 3, 4], and
with the problem of assigning labels in unknown networks [7].

Organization of the paper: In Section 2 we refine the upper bound on the
number of rounds for blind broadcasting, in the case of networks of bounded
degree. These first results are immediate conclusions from other work, however
we have to insert them for later use. Section 3 gives a non-trivial technique for
lower bounding which leads to tight results for degree up to 5. Section 4 addresses
the dependance of broadcast time on size and depth, and in Section 5 we show that
our results can also be used to improve time bounds for full synchronization in
small-degree networks.

2 Simple General Bounds

First of all, the upper bound proof of [8] can be easily extended to the case of
bounded degree. Recall that $d$ denotes the depth.

**Proposition 1** Protocol BB accomplishes blind broadcasting in networks with $n$
nodes of maximum degree $\Delta$ in at most $\min\{(\Delta - 1)d + \Delta, 2n - d\} \leq (2 - \frac{2}{\Delta})n + O(1)$ rounds.

**Proof.** The $2n - d$ bound is shown in [8], so it remains to prove the other bound.
Though it is folklore (cf. [1, 6, 9]) we include the argument: Divide the node
set into levels, according to the distance from the source. Every node except the
source has been informed through one edge, and the other edges incident to it
become saturated within the next $\Delta - 1$ rounds. Hence each but the last level
is completely informed after $\Delta - 1$ rounds, and moreover, all edges inside the
current level as well as the edges to the next level become saturated. This gives the \((\Delta - 1)d\) term. Additional \(\Delta\) rounds are required to saturate the last level.

Due to the following observation, the coefficient of \(n\) goes to 2 as \(\Delta\) grows. The proof is again an easy adaptation from [8].

**Proposition 2** Protocol BB needs at least \(2 - \frac{3}{\Delta + 1}\) rounds in the worst case, for \(\Delta \equiv 5 \mod 6\).

**Proof.** For simplicity we consider only this case, generalization to other \(\Delta\) is straightforward. Let be \(\Delta = 6k - 1\). Consider a network with \(n/2k\) levels, each consisting of a \(2k\)-clique, where neighbored levels are linked by the edge set of a complete bipartite graph. Since the edge set of a \(2k\)-clique can be partitioned into \(2k - 1\) disjoint perfect matchings, one easily sees that an adversary can defer informing the next \(2k\) nodes for \(4k - 1\) rounds.

In the next section we will get improved upper bounds, at least for \(\Delta\) up to 5. Moreover, our new bounds are tight.

### 3 Tight Bounds for Small Degrees

First we describe the overall idea of our approach. For some constant factor \(b\), we wish to prove that BB terminates in every network with \(n\) nodes and maximum degree \(\Delta\) after at most \(r := bn\) rounds. Equivalently we show that after \(r\) rounds at least \(b^{-1}r\) nodes will be informed, whatever the network topology is.

We assume for contradiction the existence of a network along with an adversary strategy such that \(n < b^{-1}r\) nodes will be informed after \(r\) rounds. Consider such a counterexample with minimum \(n\). It is convenient to number the rounds by \(1, 2, 3, \ldots\) in reverse chronological order, i.e. the \(k\)-th last round is called *round \(k\)*. For brevity, a node that gets *informed* in round \(k\) is called *a node in round \(k\)*, and an edge that becomes *saturated* in round \(k\) is called *an edge in round \(k\)*. Symbol \(x_i\) refers to a node in round \(i\), and if there are several such nodes, we denote them \(x_i, x'_i\) etc.

Since the source necessarily informs exactly one node \(x_r\) in round \(r\), we may pretend that exactly two adjacent nodes are in round \(r\), (namely the source and \(x_r\)), and that the edge joining them is in round \(r\). This will avoid case distinctions between the source and non-source nodes. For general \(i\) observe:

(a) Every node in round \(i\) is incident to some edge in round \(i\).

This follows directly from the definitions.

Now assume that, for some \(k\), at least \(b^{-1}k\) nodes are in rounds \(1, \ldots, k\). Consider the sub-network consisting of nodes in rounds greater than \(k\). If an adversary applies the same strategy in this sub-network as in the entire network but stops before round \(k\), we obviously have \((n - b^{-1}k)/(r - k) < b^{-1}\). This contradicts
minimality of our counterexample. Hence we can suppose for any positive $k$:
(b) Strictly fewer than $b^{-1}k$ nodes are in rounds $1, \ldots, k$.

Also note that protocol BB requires:
(c) Every informed node sends a message in every round during its lifetime, i.e.
before it becomes saturated.

As a consequence we get:
(d) If an edge is in round $i$ then both the sending node and the receiving node are
in one of the rounds $i + \Delta - 1, \ldots, i$.

This is clear since nodes from earlier rounds are already saturated at this mo-
moment.

The idea of our technique is to derive a contradiction from $n < b^{-1}r$ and obser-
vations (a)–(d). We demonstrate the method for $\Delta = 3, 4, 5$. The proofs are similar
to each other, but we have to be careful about different details.

**Theorem 1** Protocol BB accomplishes blind broadcasting in networks with $n$
nodes of maximum degree 3 in at most $n$ rounds, and this bound is tight.

**Proof.** Trivially, the bound of $n$ rounds (that is, $b = 1$) cannot be improved, even
for $\Delta = 2$. Now we establish the upper bound. Due to (b), no node is in round 1.
But at least one edge is in round 1, and its endnodes are in rounds 2 or 3, because
of (c) and (d). Now (b) enforces: Either both round 2 and 3 contain exactly one
node or round 3 contains exactly two nodes.

In the former case, node $x_2$ cannot be informed by $x_3$, since edge $x_2x_3$ is in
round 1. Hence $x_2$ is informed during round 2 by some other node which must
be $x_4$, due to (d). According to (c), $x_3$ also sends a message in round 2, and by
(d) again, its only possible recipient is $x_4$. Due to (c), $x_4$ also sends some message
in round 3. Moreover, (a) implies that $x_4$ is incident to some edge in round 4.
Altogether $x_4$ has already degree 4, contradiction.

In the latter case, arguments are similar: Both $x_3$ and $x_3'$ send messages to $x_4$
in round 2, thus $x_4$ gets degree 4. \hfill $\Box$

To make the next, more involved proofs more readable, we omit the pointers
to (a)–(d), as one can always easily find which observation is applied. Due to
limited space, we did not include figures.

**Theorem 2** Protocol BB accomplishes blind broadcasting in networks with $n$
nodes of maximum degree 4 in at most $\frac{5}{4}n$ rounds, and this bound is tight.

**Proof.** First we give an example of arbitrarily large networks where BB needs
$\frac{5}{4}n$ rounds in the worst case: For $n$ divisible by 4, our network is a chain of $n/4$
cliques with nodes $u_i, v_i, x_i, y_i$. These cliques are linked by edges $x_iu_{i+1}$ and $y_iv_{i+1}$,
for $1 \leq i < n/4$. Node $u_1$ is the source. It is straightforward to verify that informing 4 nodes always requires 5 rounds in the worst case.

Next we prove that $\frac{n}{5}$ rounds are sufficient. Note that $b^{-1} = \frac{4}{5}$. Thus round 1 contains no node, but some edge is in round 1, whose endpoints $u, v$ must be in rounds 2, 3, 4.

Assume that rounds 2, 3, 4 do not contain more nodes. Note that round 5 can have at most one node $x_5$. In round 2, each of $u, v$ is either informed by $x_5$ (if $u$ and $v$, respectively, belong to round 2), or it sends a message to $x_5$ (otherwise). In either case, two edges incident to $x_5$ are in round 2. Since $x_5$ is unsaturated before round 2, $x_5$ has to send messages in rounds 4 and 3, and moreover, $x_5$ is incident to some edge from round 5. Thus $x_5$ has degree at least 5, contradiction.

We conclude that exactly three nodes $u, v, w$ are in rounds 2, 3, 4, hence no node is in round 5, round 6 has at most one node $x_6$, and rounds 6, 7 together have at most two nodes. We obtain the following cases where numbers indicate the rounds $u, v, w$ belong to: $(444), (443), (433), (442), (432)$.

Recall that edge $uv$ must be still unsaturated after round 2. During round 2, nodes $u$ and $v$ can send messages only to $w$ (since there is no other possible recipient node in round 5). It follows that edges $uw$ and $vw$ are unsaturated just after round 3. In summary, all edges between $u, v, w$ are unsaturated after round 3. In particular, those of $u, v, w$ being in rounds 3 and 4 have been informed by nodes from earlier rounds, i.e., from 6 and 7. Obviously, this is impossible in cases $(444)$ and $(433)$. In cases $(443)$ and $(442)$, the two nodes from round 4 have to send messages to $x_6$ during round 3, but since one of them has been informed by $x_6$, the corresponding edge is already saturated, contradiction. It remains case $(432)$.

We denote the nodes in $\{u, v, w\}$ by $x_4, x_3, x_2$.

As we have seen, the only node that can inform $x_3$ is $x_6$. Assume that $x_4$ is still unsaturated after round 3. Then $x_4$ must have sent a message to $x_6$ in round 3, in order to keep edge $x_4x_3$ unsaturated. Moreover, $x_6$ has to send messages in rounds 5 and 4, and it is incident to some edge from round 6. Altogether it has degree at least 5, contradiction. Consequently, $x_3$ is the only unsaturated node after round 3. But now $x_3$ must inform $x_2$, and no unsaturated edge is left for round 1. This final contradiction completes the proof. □

**Theorem 3** Protocol BB accomplishes blind broadcasting in networks with $n$ nodes of maximum degree 5 in at most $\frac{3}{2}n$ rounds, and this bound is tight.

**Proof.** The lower bound follows from Proposition 2.

Note that $b^{-1} = \frac{4}{5}$. Round 1 contains no node, but some edge is in round 1, whose endpoints $u, v$ must be in rounds 2, ..., 5. Assume that these rounds do not contain other nodes than $u$ and $v$. Round 6 can have at most one node $x_6$. By the same argument as earlier, $x_6$ gets degree at least 6, contradiction.

We conclude that exactly three nodes $u, v, w$ are in rounds 2, ..., 5, round 6 contains no node, round 7 has at most one node, and rounds 7, 8, 9 together have
at most two nodes. We obtain the following cases with respect to \( u, v, w \) (using the same notation as before): (555), (554), (544), (553), (552), (542).

Again, no edges between \( u, v, w \) are saturated already after round 3. In particular, those of \( u, v, w \) being in rounds 3,4,5 have been informed by nodes from earlier rounds, i.e. from 7,8,9. Obviously, this is impossible in case (555). In the other (55*) cases, the two nodes in round 5 have to send messages to \( x_7 \) during round 3, otherwise they become saturated. But since one of them has been informed by \( x_7 \), the corresponding edge is already saturated, contradiction. Hence at most two nodes are still unsaturated after round 3, leading to a contradiction again. The remaining cases are (542), (543), and (544).

Case (542): As we already know, both \( x_4 \) and \( x_5 \) are unsaturated after round 3, therefore both of them send messages to \( x_7 \) during round 3. This violates the assumption of maximum degree 5.

Case (543): The only node that can inform \( x_3 \) is \( x_7 \). Assume that \( x_4 \) is still unsaturated after round 3. Then \( x_4 \) has sent a message to \( x_7 \) in round 3, otherwise edge \( x_4x_3 \) becomes saturated. Once more we obtain degree at least 6 for \( x_7 \), contradiction. Consequently, \( x_3 \) and \( x_5 \) are the only unsaturated nodes after round 3, contradiction.

Case (544): In round 3, at most one node from round 4 can send a message to \( x_7 \), otherwise \( x_7 \) gets degree at least 6. That means, at most \( x_5 \) and one node from round 4 are unsaturated after round 3, so BB stops before round 1, contradiction. Hence this case is also impossible.

Note that our constraints are powerful enough to finish case inspection after a constant number of rounds. Here we leave it open whether the method remains likewise successful for \( \Delta \geq 6 \). One may object that more elegant proofs are desirable. However the case distinctions seem to be unavoidable.

## 4 Broadcast Time Dependent on Size and Depth

In the preceding section we studied factor \( b \) in the worst-case broadcast time \( bn \). However Proposition 1 suggests that the broadcast time should be expressed in both \( n \) and \( d \), as linear bounds in \( n \) are poor if the depth-to-size ratio \( q := d/n \) is small. On the other hand, the question is completely settled for small \( q \) by:

**Proposition 3** The upper bound \( b = (\Delta - 1)q \) is tight if \( q \leq \frac{1}{\Delta - 1} \).

**Proof.** Consider a network where all nodes on some “critical path” \( P \) from the source to some node at distance \( d \) are incident to \( \Delta - 2 \) additional nodes, such that all these \( (\Delta - 2)d \) additional nodes are distinct and not on \( P \). More nodes may be added in a tree-like fashion. By an obvious adversary strategy, progress of broadcasting on \( P \) needs \( \Delta - 1 \) rounds per edge.
Designed networks typically have small depth and diameter (compared to $n$), ideally $d = O(\log n)$, but this cannot be assumed e.g. for “anarchistically grown” networks with unknown or varying topology. Therefore larger $q$ must also be considered if one is interested in guaranteed upper bounds for broadcasting. So let us also study the case $q \geq \frac{1}{\Delta-1}$.

Note that the simple bound from Proposition 1 is $b \leq (\Delta-1)q$ if $q \leq \frac{2}{\Delta}$, and $b \leq 2 - q$ otherwise. This table summarizes the upper bounds obtained by combining all the previous results:

<table>
<thead>
<tr>
<th>$\Delta$</th>
<th>$q \in$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$[1/2, 1)$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$[1/3, 5/12)$</td>
<td>$3q$</td>
</tr>
<tr>
<td>4</td>
<td>$[5/12, 3/4)$</td>
<td>$5/4$</td>
</tr>
<tr>
<td>4</td>
<td>$[3/4, 1)$</td>
<td>$2 - q$</td>
</tr>
<tr>
<td>5</td>
<td>$[1/4, 3/8)$</td>
<td>$4q$</td>
</tr>
<tr>
<td>5</td>
<td>$[3/8, 1/2)$</td>
<td>$3/2$</td>
</tr>
<tr>
<td>5</td>
<td>$[1/2, 1)$</td>
<td>$2 - q$</td>
</tr>
</tbody>
</table>

Now it is an obvious question to derive, for fixed $\Delta$, lower bounds for $b$ as a function of parameter $q$.

The complete answer is easy for $\Delta = 3$: Consider e.g. a path where some fraction of nodes are adjacent to a pendant node (of degree 1). Then the broadcast time is roughly $n$, while $q$ can be any value between $\frac{1}{2}$ and 1. Hence Proposition 3 together with our table provides a tight bound for any $q$.

Next we consider $\Delta = 4$.

**Proposition 4** There exist arbitrarily large networks with $\Delta = 4$ and any $q \in [\frac{1}{2}, 1)$ such that $b \geq \frac{3}{2}q + \frac{1}{2}$. Similarly, for any $q \in [\frac{1}{2}, 1)$ we have $b \geq \frac{3}{2} - \frac{1}{2}q$.

**Proof.** For simplicity we suppress constant summands in the following calculations, which is irrelevant for large $n$.

Our network witnessing the first lower bound consists of two parts: a chain of cliques of size 4 which are linked as in the proof of Theorem 2, and a caterpillar with two hairs of length 1 attached to every node of the body.

By definition of $q$, the depth is $d = qn$. Every clique contributes 2 to the depth, and every segment of the caterpillar contributes 1. Remember that 5 rounds are required to inform every clique, thus the number of rounds BB has to spend to
inform all cliques is $\frac{5}{2}$ times the depth. Similarly, 3 rounds are required to inform every segment of the caterpillar.

Since $n$ is the total number of nodes, the caterpillar has depth $(1 - 2q)n$, and thus the clique chain has depth $(3q - 1)n$. It follows

$$b = \frac{5}{2}(3q - 1) + 3(1 - 2q) = \frac{3}{2}q + \frac{1}{2}.$$

Our network witnessing the second lower bound also consists of a chain of cliques of size 4 and a path. By similar arguments as above, the clique chain has depth $(1 - q)n$, and the path has length $(2q - 1)n$. Thus we get

$$b = 2q - 1 + \frac{5}{2}(1 - q) = \frac{3}{2} - \frac{1}{2}q.$$

For $\Delta = 5$ we similarly obtain some lower bounds. Note that, neatly, we get a matching lower bound for $q \leq \frac{1}{2}$.

**Proposition 5** There exist arbitrarily large networks with $\Delta = 5$ and any $q \in \left[\frac{1}{4}, \frac{1}{2}\right]$ where $b \geq 2q + \frac{1}{2}$. For any $q \in \left[\frac{1}{2}, 1\right)$ we have $b \geq 2 - q$.

**Proof.** One part of our networks is a chain of nodes $x_i, y_i$, $i = 1, 2, 3, \ldots$, with all possible edges between nodes whose indices differ by at most 1. (Cf. Proposition 2, case $k = 1$.) Note that every pair of nodes contributes 1 to the depth, and 3 rounds are required to inform a pair.

For $q \leq \frac{1}{2}$, the other part of the network is a caterpillar with three hairs of length 1 attached to every node of the body. The caterpillar contains $(1 - 2q)n$ nodes, hence its depth is $(\frac{1}{2} - q)n$. Consequently, the chain of pairs has depth $(2q - \frac{1}{2})n$. Therefore

$$b = 3(2q - \frac{1}{2}) + 4(\frac{1}{2} - q) = 2q + \frac{1}{2}.$$

For $q \geq \frac{1}{2}$, the other part of the network is simply a path. Here the chain of pairs has depth $(1 - q)n$, and the path has length $(2q - 1)n$. Thus

$$b = 2q - 1 + 3(1 - q) = 2 - q.$$

The figures depict our knowledge about the worst-case broadcast time $bn$ for degree 4 and 5. Note that the horizontal lines are the non-obvious upper bounds from the preceding section, and that the exact $b$ remains unclear in small regions only.
Full synchronization is the following communication task. As in broadcasting, some message from the source has to reach all nodes, but additionally, at some moment which is computable by all nodes, every node must also know that all other nodes are informed, know that all nodes know that all nodes are informed, and so on. At such a moment, all nodes can simultaneously go into some termination status. It has been shown in [8] that full synchronization in arbitrary networks can be simply achieved in $3n$ rounds, and a slightly better but involved protocol has been announced. In this section we show that in networks with fixed maximum degree, full synchronization needs only $n$ rounds more than blind broadcasting. Hence Theorems 1, 2, 3 immediately yield considerably better bounds than $3n$ for full synchronization in the case of small-degree networks. The basic idea of the protocol is from [8], but we have to modify it to some extent.

Theorem 4 Full synchronization in networks with fixed maximum degree $\Delta$ can be accomplished in $(b + 1)n + O(1)$ rounds, provided that BB needs at most $bn$ rounds.

Proof. Every node $v$ behaves as follows. First, $v$ runs protocol BB. As soon as $v$ is saturated, $v$ sends a message “you are my father” to the node that informed $v$ first; ties are broken arbitrarily. In the following rounds, $v$ sends a message “you are not my father” to all other neighbors. (The source sends, of course, only messages of the latter type.) The “you are my father” messages define a rooted spanning tree in the network in an obvious sense. Also observe that every node learns, at most $\Delta$ rounds after saturation, which of its network neighbors are also tree neighbors. In particular, every node knows whether it is a leaf or an inner node in the tree, at
most $O(1)$ rounds after BB has been finished. (Here a leaf is a node with degree 1 in the tree, in particular the source can also become a leaf.)

Now the nodes ignore the orientations of edges in the tree, i.e. they consider it as an unrooted tree, and proceed further as follows. As soon as a node has learned that it is a leaf, it sends message “my subtree has size 1” to its unique tree neighbor. We say that the leaf has been activated. Every node that has received size messages with numbers $x_1, \ldots, x_k$ from all tree neighbors except exactly one, say $u$, computes $x = x_1 + \cdots + x_k + 1$ and sends message “my subtree has size $x$” to $u$.

This establishes another orientation on the edges of the tree. It is not hard to see that one of the following cases appears when the last nodes receive size messages:

1. A unique node $u$ in the tree receives simultaneously the two or more latest size messages from its tree neighborhood.
2. A unique pair of adjacent nodes $v, w$ simultaneously exchange size messages through their common edge.

Moreover, in case (1), node $u$ knows $n$, and in case (2), both $v$ and $w$ know $n$. It remains to broadcast value $n$ to all nodes via the tree edges. Next we will show that this can be done in at most $n + O(1)$ rounds after termination of BB. (This was already claimed in [8], but no proof has been given.)

Consider case (1). Let $x_1 \geq \cdots \geq x_k$ be the sizes of subtrees if we would remove $u$. (These are exactly the numbers in the size messages received by $u$.) Suppose w.l.o.g. (by shifting the time axis) that the last leaf sends its size message at time 0. Then, for any $i$, the size message containing number $x_i$ reaches $u$ before time $x_i$: The last leaf in the corresponding subtree sends at time 0 or before, and after that, in every round, at least one further node in the subtree sends its size message. The crucial observation is: Since the two or more latest size messages get in simultaneously, $u$ has all size messages at time $x_2$ or earlier. In the subsequent broadcasting phase, $u$ sends $n$ to its tree neighbors according to decreasing $x_i$. Broadcasting in the $i$th subtree requires at most $x_i$ rounds, by the same trivial argument as above. Hence broadcasting is finished after at most $\max x_i + i$ rounds. It follows quite easily that the total time after activation of the last leaf, and hence after termination of BB, is bounded by $x_1 + x_2 + O(1) \leq n + O(1)$.

Case (2) is resolved essentially in the same way, where the pair $\{v, w\}$ takes over the role of $u$. Note especially that the two latest size messages from outside $\{v, w\}$ reach $v$ and $w$ simultaneously, thus the same counting argument works.

Thus we have shown that all nodes can learn the network size $n$ within the claimed number of rounds. In order to achieve full synchronization, the nodes must also learn the time when the protocol has been started, and the time $bn$ when BB has been finished. Then they can wait until round $(b + 1)n + O(1)$ after the start of BB and go into the termination status. But this is simply to manage by an obvious enhancement of the above protocol: The start time can be broadcast in the BB phase, as an attachment to the message. Along with the size messages,
the leaves send time stamps indicating the time when they have been informed, and every node receiving size messages computes the maximum of received time stamps. Then \( u \) (or \( v, w \)) also learns the termination time of BB and can broadcast it along with \( n \).

\[ \square \]

References


*Peter Damaschke* is with the School of Mathematical and Computing Sciences at Chalmers University of Technology, Göteborg, Sweden. E-mail: ptr@cs.chalmers.se
The Client-Server 2-Spanner Problem and Applications to Network Design

(Extended abstract)

MICHAEL ELKIN
The Weizmann Institute of Science, Israel

DAVID PELEG
The Weizmann Institute of Science, Israel

Abstract
This paper studies the client-server version of the 2-spanner problem. This is a natural generalization of the sparse 2-spanner problem that turns out to be useful for certain network design problems. The paper presents a logarithmic ratio approximation algorithm for this problem. The algorithm and the analysis are then extended to the directed client-server 2-spanner problem and to the client-server augmentation 2-spanner problem, establishing a logarithmic approximation ratio for these problems too.

The paper also studies the bounded-diameter network design (BDND) problem, and establishes a logarithmic approximation ratio for the problem, using the tools developed for the client-server 2-spanner problem.

1 Introduction

1.1 Motivation
Consider the following type of network design problem, which we refer to as bounded-diameter network design (BDND). We are given a set $V$ of $n$ vertices, and a set of connection requests $R = \{(s_i, d_i) \mid s_i, d_i \in V, \ 1 \leq i \leq n\}$. Our task is to construct a network for connecting each request pair in $R$ by a path, subject to two additional constraints. First, not every two sites can be connected by a direct link. Rather, we are also given a set of potential links $T \subseteq V \times V$, and the network may only use links from the set $T$. Secondly, we are also given a diameter bound $k$, and the connection paths are required to consist of at most $k$ links. A feasible solution to the problem thus consists of a network $H(V, E)$ connecting the $n$ sites, such that $E \subseteq T$, and for each request pair $(s_i, d_i) \in R$, there exists a connecting path in $H$ of $j$ edges, where $j \leq k$,

\[
(s_i, v^1), (v^1, v^2), \ldots , (v^{j-2}, v^{j-1}), (v^{j-1}, d_i) \in H. \tag{1}
\]
The $k$-BDND problem is to find the sparsest feasible network $H$ possible, where sparsity is measured in terms of the number of links used, $|E|$.

A natural idea would be to approach this problem using the concept of $k$-spanners. Indeed, graph spanners have been proposed in the past as a basic tool for handling network design problems [9, 11]. Given a graph $G = (V, E)$, for every pair of vertices $u, v \in V$, let $P(u, v, G)$ be the set of all the simple paths from $u$ to $v$ in $G$. Letting $|P|$ denote the length of the path $P$, define the distance between $u$ and $v$ in $G$ as $\text{dist}_G(u, v) = \min_{P \in P(u, v, G)} |P|$. A subgraph $H = (V, E')$ is a $k$-spanner for $G$ if $\text{dist}_H(u, v) \leq k$ for every edge $(u, v)$ in $E$. The sparsest $k$-spanner problem is the problem of finding a $k$-spanner $H$ of the given graph with a minimum number of edges.

Viewing $G$ as a logical graph whose edges represent node pairs requiring communication, a sparse spanner provides a cheap design for a network capable of meeting the requirements and guaranteeing short connections (i.e., with a small number of hops).

An evident obstacle on the way to applying this approach is that both the definition of spanners and the existing algorithms for constructing sparse spanners make no distinction between the two potentially different roles of edges in the logical graph $G$, namely, as representing “client pairs” (node pairs that need to communicate) and “servers pairs” (node pairs that can be connected by a link). Specifically, these formulations require $H$ to (1) be a subgraph of $G$, and (2) have a short path corresponding to every edge of $G$. Clearly, there are natural situations (including, in particular, the BDND problem) in which it is necessary to distinguish between these two functions, and identify two distinct sets of node pairs, the set $C$ of client edges, which are the ones required to be covered, and the set $S$ of server edges, which can be used as covering edges, where neither set is necessarily contained in the other.

This paper deals with approximation algorithms for the general “client-server” variant of the 2-spanner problem, and presents a number of example applications for this problem in the context of network design.

### 1.2 The Client-Server 2-Spanner Problem

The client-server version of the 2-spanner problem (henceforth denoted $C \cdot S$ 2-Spanner) is a generalization of the sparsest 2-spanner problem. By the above definition, a 2-spanner of a graph $G = (V, E)$ is a subset $H$ of the edge set of the graph, such that for any edge $e = (u, v)$ in $G$, either there are two edges of $H$ (called the spanning or covering edges for $e$) which create a length 2 path from $u$ to $v$, or $e$ itself is in $H$ (in which case we say that $e$ is self-covered).

The client-server version of the problem is formally defined as follows. We are given a graph $G = (V, E)$ and two edge sets $C$ and $S$ called the client edge set and server edge set respectively. An $S$ 2-spanner for $C$ is a set $H \subseteq S$ that covers all edges of $C$. Given $G$, $S$ and $C$, we look for the sparsest $S$ 2-spanner
Clearly, the original 2-spanner problem is a special case of the C-S 2-spanner problem where both $C$ and $S$ are equal to $E$. It immediately follows from this observation that the client-server version of the 2-spanner problem is NP-complete, because the 2-spanner problem is NP-complete [12].

### 1.3 Our Results

We devise an $O(\log \frac{1}{|V(C)|})$-approximation algorithm for the C-S 2-spanner problem. The algorithm is based on the ideas of the greedy algorithm of [8] for the 2-spanner problem, but the need to distinguish the client and server sets complicates both the algorithm and its analysis.

We then extend this algorithm to the directed C-S 2-spanner problem and show that it admits the same approximation ratio. Then we consider C-S augmentation problem. It is a somewhat more general variant of the C-S 2-spanner problem, where it is assumed that a spanner $H$ for $C$ is already partially constructed, and the goal is to cheaply complete $H$ into a 2-spanner for $C$. We present a variant of our algorithm providing an $O(\log \frac{1}{|V(C)|})$-approximation for this problem too.

Let us remark that the client-server version of the 2-spanner problem can be reduced to the unit-length weighted 2-spanner problem, which admits an $O(\log n)$ approximation [7]. However, the approximation algorithm presented in the current paper gives $O(\log \frac{1}{|V(C)|})$ approximation ratio, which is considerably better than $O(\log n)$ for typical applications, because the requirements collection $C$ is typically sparse with respect to $V(C)$, and the ratio $|C|/|V(C)|$ may often be bounded by a small constant. Also, our algorithm treats the directed case, that is needed for some of our later applications to network design. Also, the techniques developed in this paper were found useful for construction of approximating algorithms for the (directed and undirected) C-S $k$-spanner problem for $k \geq 2$ [6].

Turning to applications in the context of network design, we first observe that the unrestricted version of the $k$-BDND problem, which we call the $k$-UBDND problem, is solvable in polynomial time. We devise an $O(\log \frac{|L|}{\pi})$-approximation algorithm for (ordered or unordered) 2-BDND problem.

We next consider hardness of approximation, and show that it is NP-hard to approximate the $k$-BDND problem with approximation ratio better than $O(\log n)$ for $k \geq 2$, i.e., there is a constant $c$ such that this problem cannot be approximated with ratio $\rho < c \log n$ unless $NP = P$. For the BDND problem with diameter $k$ we show that for $k > 2$ it is $2^{2\log n^c}$-inapproximable for any $0 < c < 1$ unless $NP = P$ (this property is henceforth termed strong inapproximability).

We also consider a generalization of network design problem in which we are given also the set $L$ of edges that are already built and we may use them for free. Hence the goal is to minimize the number of new edges used in the design. We call this problem $k$-BDND augmentation problem. We show that the 2-BDND augmentation problem, in all presented above versions (ordered/unordered,
restricted/unrestricted) admits $O(\log \frac{\log n}{n})$-approximation ratio. Another restricted version of the BDND problem that we analyze is the self-contained $k$-BDND (or simply $k$-SBDND) problem, defined in Section 6. We prove that already self-contained $k$-BDND augmentation problem is both NP-hard for approximation with ratio better than $O(\log n)$ for $k \geq 2$, and strongly inapproximable for $k > 3$. Furthermore, with the order restriction the hardness result applies for $k = 3$ too.

An even more general version of the $k$-BDND problem is the one when each pair $(a_i, a_j)$ is assigned a cost, specifying how expensive it is to build an edge between these two particular nodes. We are interested in minimizing this cost. We devise an $O(\log n)$ approximation algorithm for this problem in case $k = 2$.

In this extended abstract we describe the most basic version of our approximation algorithm for the C-S 2-spanner problem and provide some intuition of its analysis. The details of the analysis, as well as its generalizations to more complicated versions of the client-server problem are omitted. Next, we sketch the proofs of our results concerning the BDND problem.

2 Definitions

Without loss of generality, assume that $E = C \cup S$.

For every edge subset $S \subseteq E$ define the neighborhood of $v$ in $S$ as $N(v, S) = \{u \mid (u, v) \in S\}$. The neighborhood of $v$ with respect to the whole set $S$ is defined as $N(v) = N(v, S)$.

The algorithm constructs the spanner graph denoted by $H$. We define two functions $C: 2^E \to 2^C$ and $U: 2^E \to 2^C$ as follows.

$$C(H) = \{(t, z) \in C \mid (t, z) \in H \text{ or } \exists v \in V \text{ s.t. } (t, v), (v, z) \in H\},$$

$$U(H) = C \setminus C(H).$$

Intuitively, $C(H)$ is the subset of client edges covered by $H$, whereas $U(H)$ is the subset of client edges that are still uncovered. Both functions are monotone; $C$ is monotonically increasing, i.e. $A \subseteq B$ implies $C(A) \subseteq C(B)$, and therefore $U$ is monotonically decreasing, i.e. $A \subseteq B$ implies $U(B) \subseteq U(A)$.

The algorithm proceeds in steps. In each step, the algorithm picks a vertex $v$ and a subset of its neighbors, $Q \subseteq N(v, S)$, such that the edges connecting $v$ to the vertices of $Q$ are server edges that have not been picked into $H$ so far, and adds these edges to $H$. We refer to this set of edges as the added edges, and define it formally as follows. For a set $S \subseteq E$, let $AE(Q, v, S) = \{z \in S \mid z \in Q\}$. Note that this definition is slightly more general, and applies also to sets $Q$ containing some nodes $z$ such that $(z, v)$ is already in $H$. The set of edges actually added to $H$ when connecting $v$ to the vertices of $Q$ is in fact $AE(Q, v, S \setminus H)$.

Adding the set of edges $AE(Q, v, S \setminus H)$ to the spanner $H$ causes the covering of some new edges that were not covered before. The set of those covered edges is
denoted by $CE(Q,v,H)$. These edges are partitioned into four disjoint classes, according to the way they are covered. Specifically, $CE(Q,v,H) = \bigcup_{i=1}^{4} CE_i(Q,v,H)$, and

$$
\begin{align*}
CE_1(Q,v,H) &= \{(t,z) \in U(H) \mid t, z \in Q\}, \\
CE_2(Q,v,H) &= \{(t,z) \in U(H) \mid z \in Q, t \in V \setminus Q, (t,v) \in H\}, \\
CE_3(Q,v,H,S) &= \{(v,t) \in U(H) \cap S \mid t \in Q\}, \\
CE_4(Q,v,H) &= \{(v,t) \in U(H) \mid t \in V \setminus Q, \exists z \in Q, (z,t) \in H\}.
\end{align*}
$$

The first set consists of edges covered by two edges with common endpoint $v$ and two other endpoints from the set $Q$, the second set consists of edges that are covered by one edge from $H$ and one edge from $v$ to a vertex of $Q$, the third set consists of self-covered edges, namely, edges that are both client and server edges with $v$ as one endpoint and a vertex from $Q$ as another, and the fourth set consists of edges that are covered by one edge from $H$ and one edge newly added to $H$ (exactly like $CE_2$), but the common vertex of these two covering edges is not $v$.

The definitions imply that these four sets are necessarily disjoint, for any fixed set $Q$. On the other hand, for different choices of $Q$, the same edge may occur in a different class $CE_i$.

Figure 1 illustrates the four types of covered edges with $e_i \in CE_i(Q,v,H)$ for every $1 \leq i \leq 4$. In addition, the edge $e'_1$ illustrates a different type of edge belonging to $CE_1$. However, note that when $Q \subseteq N(v,S)$, there are no edges of $H$ that connect $v$ to some node of $Q$. Therefore, in this case no edges of type $e'_1$ may appear in $CE_1$.

We use small letters to denote set sizes, e.g., the size of the set $AE(Q,v,S)$ is denoted by $ae(Q,v,S)$.

We define the **density** of the choice of $Q \subseteq N(v,S \setminus H)$ as the covered-to-added edge ratio,
\[ \tilde{\rho}(Q, v, H) = \frac{ce(Q, v, H)}{ae(Q, v, S \setminus H)}. \]

If \( ae(Q, v, S \setminus H) = 0 \) then necessarily \( Q = \emptyset \) and so \( ce(Q, v, H) = 0 \), hence we define \( \tilde{\rho}(Q, v, H) = 0 \) in this case.

The maximum density for a node \( v \) with respect to a partial spanner set \( H \) is

\[ \rho(v, H) = \max_{Q \subseteq N(v, S(H))} \{ \tilde{\rho}(Q, v, H) \}. \]

We also define another notion of covered edges. The set of edges that are covered in the current step entirely by the edges added to \( H \) in this step, without using edges already existing in \( H \), is defined as

\[
\begin{align*}
COV(Q, v, H, S) &= CE_1(Q, v, H) \cup CE_2(Q, v, H, S), \\
COV(Q, v, H) &= COV(Q, v, H, S).
\end{align*}
\]

(The only exception is that edges such as \( e'_1 \) in Fig. 1 do belong to \( CE_1 \), hence to \( COV \), even though they are spanned with the help of \( H \).) Using this notion we can talk about a different type of density, referred to as \( \varphi \)-density. This density is defined for \( Q \subseteq N(v, S) \), for any subset \( S \subseteq S \), as follows:

\[
\begin{align*}
\varphi(Q, v, H, S) &= \frac{cov(Q, v, H, S)}{ae(Q, v, S)}, \quad \varphi(Q, v, H) = \varphi(Q, v, H, S), \\
\varphi(v, H, S) &= \max_{Q \subseteq N(v, S)} \{ \varphi(Q, v, H, S) \}, \\
\varphi(v, H) &= \varphi(v, H, S).
\end{align*}
\]

In the definition of \( \varphi \), if \( ae(Q, v, S) = 0 \) then again necessarily \( Q = \emptyset \) (since \( Q \subseteq N(v, S) \)) and so \( \varphi(Q, v, H, S) = 0 \). Hence in this case we again define \( \varphi(Q, v, H, S) = \rho(Q, v, H) = 0 \).

Finally let \( COV(v, H) = COV(Q_v, v, H) \), where \( Q_v \) is the maximizing set satisfying \( \varphi(v, H) = \varphi(Q_v, v, H) \). Analogously, let \( CE(v, H) = CE(Q_v, v, H) \), for the set \( Q \) satisfying \( \rho(v, H) = \tilde{\rho}(Q, v, H) \).

Intuitively, the difference between \( \rho \) and \( \varphi \) is that the \( \varphi \)-density is an "independent density measure" of each vertex, which does not depend on \( H \) and hence does not reflect our progress in building the spanner. In particular, the definition of \( \varphi \)-density underestimates the possible gain from connecting \( v \) to \( Q \), as it ignores the aid of the existing \( H \), and at the same time, it overestimates the cost of connecting \( v \) to \( Q \), as it charges also for edges already in \( H \).

A useful property of \( \varphi \)-density \( \varphi(v, H) \), which our algorithm exploits, is that it decreases monotonically as \( H \) as built through the algorithm.

The \( \rho \)-density of a vertex \( v \), on the other hand, depends on the existing \( H \) and reflects the ratio between actual number of new edges that will be covered using \( v \).
and the actual number of edges that we will add choosing \( v \) at the current stage of the algorithm. This density does not change monotonically (in particular, it might increase). However, it has another nice property, namely, that the sets of covered and added edges that are built for calculating the value of \( \rho \) at different iterations of our algorithm are disjoint even when the same vertex is chosen at two different iterations. We combine these properties throughout our analysis.

3 The Algorithm

The algorithm is a modification of 2-spanner approximation algorithm devised in [8]. Its main part is a loop, that continues while there is at least one vertex with positive \( \rho \)-density. Note that although \( \rho \)-density is used for the exiting condition of the loop, we could just as well use \( \phi \)-density, since there exists some \( v \in V \) with \( \rho(v, H) > 0 \) if and only if there exists some \( u \in V \) with \( \phi(u, H) > 0 \). The proof of this claim is omitted from this extended abstract.

Inside the loop, we pick a vertex \( v \) that maximizes the \( \phi \)-density, and compute for it the maximum \( \rho \)-density and the set \( Q_v \) yielding it. This alternation between the \( \rho \)-density and the \( \phi \)-density deserves some explanation. Basing the choice of \( v \) on maximizing the \( \phi \)-density allows us to rely on monotonicity of \( \phi \), which is useful for the analysis. On the other hand, the choice of the edges to be added to \( H \) must be based on the \( \rho \)-density, which enables us to add to the set of covered edges \( H^c \) all edges that were actually spanned during the iteration and to add to \( H \) only the edges that are needed for spanning \( H^c \) edges. (Note that in fact we do not need the value of \( \rho \) inside the loop, but rather only the \( CE \) and \( AE \) sets; however, constructing these sets is equivalent to calculating \( \rho \).)

The algorithm is described formally next. The only nontrivial component which is left unspecified for now is the computation of the \( \phi \) and \( \rho \)-densities. These are discussed in sections 3.1 and 3.2.

Algorithm \( \Lambda \)

1. \( H^u \leftarrow C; \ H^c \leftarrow \phi; \ H \leftarrow \phi \)
2. While \( \exists v \) s.t. \( \rho(v, H) > 0 \) do :
   (a) Choose vertex \( v \) that maximizes \( \phi(v, H) \)
   (b) Compute \( \rho(v, H) \); let \( Q_v \) be the densest subset of \( N(v) \)
   (c) \( H^c \leftarrow H^c \cup CE(Q_v, v, H); \ H \leftarrow H \cup AE(Q_v, v, S \setminus H) \)
   (d) \( H^u \leftarrow H^u \setminus H^c \)
3. If \( H^u = \phi \) then return(\( H \)) Else return(“No 2-spanner exists”)

Remark: The sets \( H^u \) and \( H^c \) are used in the algorithm to store the values of \( U(H) \) and \( C(H) \) respectively. These sets are constructed in an incremental fashion, rather
than by directly applying functions \( U \) and \( C \) to the current \( H \) in each step, for efficiency reasons only: computing \( C(H) \) and \( U(H) \) directly would increase the running time of the algorithm. Note that the definitions of \( CE(Q,v,H) \), \( p(v,H) \) and \( \phi(v,H) \) involve \( U(H) \), hence during the algorithm we use \( H' \) instead.

### 3.1 Computing the Densities

We reduce the problems of density computation to the provisioning problem, formulated as follows.

**Input:** A collection of \( n \) items \( \{1, \ldots, n\} \) with costs \( c_j > 0 \) for \( j \in \{1, \ldots, n\} \), and \( m \) subsets of items \( S_1, \ldots, S_m \) with benefits \( b_1, \ldots, b_m \), where \( S_i \subseteq \{1, \ldots, n\} \) for \( 1 \leq i \leq m \) and \( b_i > 0 \).

**Solution:** A subset \( R \subseteq \{1, \ldots, n\} \). The cost of the solution \( R \) is \( \sum_{j \in R} c_j \). The benefit of \( R \) is the sum of benefits of subsets for which all their items were taken to the solution set, i.e.

\[
\sum_{j=1}^{m} b_j \cdot I_j(R), \text{ where } I_j(R) \text{ is defined as}
\]

\[
I_j(R) = \begin{cases} 
1 & \text{ iff } S_j \subseteq R \\
0 & \text{ iff } S_j \nsubseteq R.
\end{cases}
\]

**Objective:** Maximize the difference \( D \) between the total benefit and the total cost of the solution.

This problem can be solved in polynomial time [10]. Let the \( \phi \)-decision problem be the problem of deciding whether \( \phi(v,H) \geq k \), given a graph \( G = (V,E) \), a vertex \( v \in V \), edges subsets \( H \) and \( S \) such that \( H \subseteq S \subseteq E \) and an integer \( k \). Clearly, the problem of computing \( \phi(v,H) \) is reducible to the \( \phi \)-decision problem, because a binary search can be conducted over all possible values of \( \phi(v,H) \).

The denominator of the density \( \phi(v,H) \) is bounded by \( n = |V| \), the numerator by \( m = |E| \) and so \( \phi(v,H) \) can be computed using at most \( O(\log(n \cdot m)) = O(\log n) \) calls to a subroutine for the \( \phi \)-decision problem.

Now we reduce the \( \phi \)-decision problem to the provisioning problem in the following way.

Let \( \mathcal{N}(v,S) \) be the set of items, and let their costs be \( c_j = k \). Let the subsets be

\[
\begin{align*}
\{ \{u\} \mid u \in \mathcal{N}(v,S) \text{ and } (v,u) \in U(H) \} & \text{ and } \\
\{ \{u,w\} \mid u,w \in \mathcal{N}(v,S) \text{ and } (u,w) \in U(H) \},
\end{align*}
\]

and let their benefits be \( b_i = 1 \).

The intuition is that for every covered edge we gain one benefit unit. We also gain 1 for every edge between \( v \) and its neighbor, if this edge is both a server and a client.

The provisioning problem will now maximize

\[
D = B - C = \text{cov}(Q,v,H) - k \cdot \ae(Q,v,S),
\]
where $Q$ is a subset of the items purchased (namely, the nodes taken into the densest subset). Now we compare $D$ with 0 and answer “yes” iff $D > 0$. Indeed, this condition is equivalent to $\varphi(v, H) > k$.

Similarly, for computing $\rho(v, H)$ we reduce it to the $\rho$-decision problem. The $\rho$-decision problem is defined analogously to the $\varphi$-decision problem. The reduction of the $\rho$-decision problem to the provisioning problem is as follows. Define $S_0 = \{ e \in S \setminus H \mid v \in e \}$. An instance of the provisioning problem is constructed as follows.

Set the collection of items to be $N(v, S_0)$, with costs $c_j = k$. Define the subsets to be

$$\{ \{u\} \mid u \in N(v, S_0) \} \text{ and } \{ \{u, w\} \mid u, w \in N(v, S_0) \text{ and } (u, w) \in U(H) \},$$

with benefits $b(\{u, w\}) = 1$ and

$$b(\{u\}) = \{\{(u, v)\} \cap U(H) \cup \{\{u, z\} \in U(H) \mid (z, v) \in H\} \cup \{\{u, z\} \in H \mid (z, v) \in U(H)\}\}.$$

The provisioning problem will maximize $D = ce(Q, v, H) - k \cdot ae(Q, v, S \setminus H)$, where $Q$ is the subset of items purchased, i.e., the subset of $v$’s neighbors, such that the edges connecting them to $v$ would be taken to the spanner. The specification that $(z, v) \in H$ ensures also that $z \not\in N(v, S_0)$ and hence no edge is counted twice. Again, by comparing $D$ to 0 we obtain the answer to $\rho$-decision problem.

### 3.2 Approximating the Densities

We later show that it may suffice to approximate the densities instead of computing them exactly. Now we discuss how this can be done more efficiently than the precise computation.

First we devise a procedure that given a number $k > 0$ checks whether $\varphi(v, H) \geq k$.

For every subsets $Z \subseteq V$ and $B \subseteq E$ and vertex $u \in Z$, define

$$\text{deg}(u, Z, B) = |\{w \in Z \mid (u, w) \in B\}|.$$  \hfill (4)

Assume $\varphi(v, H) \geq k$. Now for every $u \in N(v)$ if $\text{deg}(u, Q_v \cup \{v\}, H^u) \leq k - 1$, then $u$ can not be contained in the densest subset $Q_v$ of $N(v)$, because deleting $u$ from $Q$, decreases the denominator of $\varphi$ by 1 and the numerator by $k - 1$ or less, and so if $\varphi(v, H) \geq k$ then $\varphi(Q_v \setminus \{u\}, v, H) > \varphi(Q_v, v, H)$, contradicting the maximality of $\varphi(Q_v, v, H)$.

The procedure checking whether $\varphi(v, H^u) \geq k$ utilizes the above observation, and works as follows:

**Procedure** $\text{Proc}(v, H, k)$
Initialize \( Q = N(v) \).
Iteratively find \( u \in Q \) s.t. \( \text{deg}(u, Q \cup \{v\}, H^u) \leq k - 1 \) (if exists) and eliminate it from \( Q \).

If the procedure terminates with an empty set \( Q \), we conclude that \( \varphi(v, H) < k \). This is because if \( \varphi(v, H) \geq k \) then there is some \( Q \subseteq N(v) \) s.t. \( \varphi(Q, v, H) = \varphi(v, H) \geq k \), that is \( Q \) consists of the vertices \( u \), satisfying for every superset \( Z \) of \( Q \) that \( \text{deg}(u, Z \cup \{v\}, H^u) \geq k \), by previous considerations, solely. So these vertices will not be eliminated during the run of the algorithm. Else we have found a set \( Q \) such that \( \text{deg}(u, Q, \cup \{v\}, H^u) \geq k \) for every \( u \in Q \) and so \( \varphi(Q, v, H) \geq \frac{k}{2} \).

Thus by conducting a binary search over the possible integer values of \( k \) (recall that there at most \( m \cdot n \) possible values of the \( \varphi \)-density) we obtain in polynomial time a set \( Q \subseteq N(v) \) with density \( \varphi(Q, v, H) \geq \frac{k}{2} \cdot \max_{Q \subseteq N(v)} \{ \varphi(Q, v, H) \} \geq \frac{1}{2} \cdot \varphi(v, H) \) and so we have a 2-approximation algorithm for \( \varphi \)-calculation.

To approximate \( \rho \) we use the same algorithm, but instead of the previously used \( \text{deg}(u, Z, B) \), we use another notion of degree, namely:

\[
\text{deg}(u, Z, B, S, v) = \left| \{ w \in Z \mid (u, w) \in B \} \cup \{ w \in V \setminus Z \mid (v, w) \in S, (u, w) \in B \} \cup \{ w \in V \setminus Z \mid (u, w) \in S \} \right| .
\]

We use this degree definition by comparing \( \text{deg}(u, Q, \cup \{v\}, H^u, H, v) \) with \( k \). By incorporating this definition of degree in the above procedure we obtain a 2-approximation algorithm for \( \rho \)-density.

### 4 Algorithm Correctness and Termination

In this section we sketch the analysis of Algorithm A. We start with proving that using the sets \( H^c \) and \( H^u \) instead of \( C(H) \) and \( U(H) \) in the algorithm does not affect its correctness.

**Lemma 1** After each iteration, \( H^c = C(H) \) (and therefore also \( H^u = U(H) \)).

**Proof.** By induction on the iterations. In each iteration \( j \), the current spanner \( H_j \) is augmented with a set of edges \( AE \) connecting \( v \) to the nodes of \( Q_v \), yielding a new spanner \( H_{j+1} = H_j \cup AE \), and the current set of covered edges \( H^c_j \) is augmented with the corresponding set of covered edges \( CE = \bigcup_{i=1}^{j} CE_i \), yielding \( H^c_{j+1} = H^c_j \cup CE \). To prove that \( H^c_{j+1} = C(H_{j+1}) \), we rely on the fact that \( C(H_{j+1}) \) is composed of three types of edges, \( C(H_{j+1}) = C_1 \cup C_2 \cup C_3 \), where \( C_1 = C(H_j) \), namely, the edges already covered by \( H_j \), \( C_2 \) consists of the edges covered entirely by \( AE \), and \( C_3 \) consists of edges covered one edge of \( H_j \) and one edge of \( AE \). By the inductive hypothesis, \( H^c_j = C_1 \). (For the first iteration, both \( H^c \) and \( C_1 \) are clearly empty.) Also, by a straightforward case analysis, we find that \( C_2 \) consists of precisely the edges of \( CE_1 \cup CE_3 \), and that \( C_3 = CE_2 \cup CE_4 \). (Again, in the first
Proof. Suppose the algorithm outputs some $H$, contradicting the assumption. By the previous lemma of the algorithm some vertex $z$ s.t. $\phi(z, H) > 0$, so the algorithm outputs “no 2-spanner exists”. Then suppose that a 2-spanner exists and suppose for contradiction that the following lemma shows that the remaining case is

Next, we prove that Algorithm A has the soundness property. Specifically,

Let $v$ be a vertex chosen in some iteration and let $H$ be the set of spanner edges before this iteration. Then $\rho(v, H) > 0$ implies that there exists some vertex $z$ s.t. $\phi(z, H) > 0$.

Proof. Let $Q_v$ be the set satisfying

$$
\rho(v, H) = \bar{\rho}(Q_v, v, H) = \frac{ce(Q_v, v, H)}{ae(Q_v, v, S \setminus H)},
$$

Next, we prove the completeness property of the algorithm. Specifically,

Lemma 3 If an $S$ 2-Spanner for $C$ exists, then the algorithm will find one.

Proof. Suppose that a 2-spanner exists and suppose for contradiction that the algorithm outputs “no 2-spanner exists”. Then $H = \emptyset$ upon exiting the loop. But since the algorithm has left the loop, $\rho(v, H) = 0$ holds for every $v \in V$.

Let $e = (u, w) \in H \subseteq C$, after leaving the loop. Consider two cases. If $e \in S$ then, noting that $e \in C$, $e \notin H$, and $w \in N(u)$, we have $e \in CE_2(N(u), u, H)$, and so $\rho(u, H) > 0$, contradiction.

The remaining case is $e \notin S$. Since $C$ has an $S$ 2-Spanner, there exist edges $e_1 = (u, v), e_2 = (v, w) \in S$ that span $e$. If both $e_1, e_2 \in H$ then $e \in C(H)$, and by Lemma 1, $e \notin H$ contradicting the assumption. So the only remaining possibility is that $e_1 \notin H$ or $e_2 \notin H$ at the end of the loop. Without loss of generality, suppose $e_1 \notin H$.

If $e_2 \notin H$ too, then letting $Q = \{u, w\}$ we have $e \in CE_1(Q, v, H)$, implying $\rho(v, H) \geq \bar{\rho}(Q, v, H) > 0$, contradicting the assumption.

On the other hand, if $e_2 \in H$ then for $Q = \{u\}$ we have $e \in \{(t, z) \in H^w \mid z \in Q, t \in V \setminus Q \text{ s.t. } (t, v) \in H\} = CE_2(Q, v, H)$. Therefore $\rho(v, H) > 0$, again contradicting the assumption.

The following lemma shows that the $\rho$-density in the stopping condition of the loop of Algorithm A can be replaced by the $\phi$-density. Formally,

Lemma 4 Let $v \in V$ be a vertex chosen in some iteration and let $H$ be the set of spanner edges before this iteration. Then $\rho(v, H) > 0$ implies that there exists some vertex $z$ s.t. $\phi(z, H) > 0$. 

Proof. Let $Q_v$ be the set satisfying

$$
\rho(v, H) = \bar{\rho}(Q_v, v, H) = \frac{ce(Q_v, v, H)}{ae(Q_v, v, S \setminus H)},
$$

iteration, $C_3, CE_2, CE_4$ are all empty.) It follows that

$$
C(H_{j+1}) = C_1 \cup C_2 \cup C_3 = H_{j+1} \cup (CE_1 \cup CE_3) \cup (CE_2 \cup CE_4) = H_{j+1} \cup CE = H_{j+1}.
$$
and consider $CE(Q_v, v, H)$. By the assumption that $\rho(v, H) > 0$, it follows that $ce(Q_v, v, H) > 0$. In order to prove the lemma we need to show that $cov(z, H) \neq 0$ for some $z$. Let $\hat{Q}_v = \{t, z : (t, z) \in CE_2(Q_v, v, H)\}$ and $P_v = \hat{Q}_v \setminus Q_v$. First we show that the set

$$Z = \bigcup_{1}^{3} CE_i(Q_v, v, H)$$

satisfies

$$Z \subseteq COV(\hat{Q}_v, v, H).$$

Indeed, since $Q_v \subseteq \hat{Q}_v$, both $CE_1(Q_v, v, H) \subseteq CE_1(\hat{Q}_v, v, H)$ and $CE_3(Q_v, v, H) \subseteq CE_3(\hat{Q}_v, v, H)$ hold, hence

$$CE_1(Q_v, v, H) \cup CE_3(Q_v, v, H) \subseteq COV(\hat{Q}_v, v, H).$$

Now consider an edge $e = (u_1, u_2) \in CE_2(Q_v, v, H)$. Without loss of generality, $u_1 \in Q_v$. Then $u_2 \in P_v$, and hence $u_1, u_2 \in \hat{Q}_v$, implying $e \in CE_1(\hat{Q}_v, v, H) \subseteq COV(\hat{Q}_v, v, H)$, and (5) is satisfied.

It follows that if $Z \neq \phi$ then $cov(\hat{Q}_v, v, H) > 0$, and hence $\Phi(v, H) \geq \Phi(\hat{Q}_v, v, H) > 0$, proving the conclusion of the lemma.

Therefore it remains to consider the case of $Z = \phi$. In this case $COV(v, H)$ may be empty, but we will show that then another vertex $z$ must exist for which $COV(z, H) \neq \phi$. Indeed as $CE(Q_v, v, H) \neq \phi$, necessarily $|CE_3(Q_v, v, H)| > 0$ and by definition of $CE_4$, there exist some nodes $z \in Q_v$ and $t \in V \setminus Q_v$ such that

$$(v, t) \in H^u$$

and $$(z, t) \in H.$$ (6)

Note that necessarily $(v, z) \notin H$, since otherwise $(v, t)$ is already spanned by $H$ (specifically by edges $(v, z)$ and $(t, z)$), contradicting the fact that $(v, t) \in H^u$.

Then let $Q_z = \{v, t\}$, we have that $(v, t) \in CE_1(Q_z, z, H)$ by (6), hence $(v, t) \in COV(Q_z, z, H)$. In addition, $(v, z) \in AE(Q_z, z, S)$.

Therefore both $cov(Q_z, z, H) > 0$ and $ae(Q_z, z, S) > 0$, implying that

$$\Phi(z, H) \geq \Phi(Q_z, z, H) = \frac{cov(Q_z, z, H)}{ae(Q_z, z, S)} > 0,$$

and we are done. $\square$

We next state without proof several other properties of the $\rho$- and $\Phi$-densities. These properties are needed for the analysis of the approximation ratio of Algorithm A. The proofs of these properties can be found in [5].

Lemma 5

1. For every $v \in V$, $\rho(v, H) \geq \Phi(v, H)$.

2. For every $H_1 \subseteq H_2$, vertex $v \in V$ and subset $S \subseteq S$, $\Phi(v, H_1, S) \geq \Phi(v, H_2, S)$. 
3. The density $\phi(v, H_j)$ monotonely decreases as the algorithm progresses.

4. For every $S_1 \subseteq S_2 \subseteq S$, vertex $v \in V$ and $H \subseteq S$, $\phi(v, H, S_1) \leq \phi(v, H, S_2)$.

Based on these properties, the following theorem can be proved. Its proof is omitted from this extended abstract (see [5]).

**Theorem 1** Algorithm A is an approximation algorithm for C-S 2-spanner problem with ratio $O(\log \frac{|C|}{|\tau(C)|})$ and running time of $O(m^2 \cdot n \cdot \log n)$.

5 **Extensions to Directed and Augmentation Variants**

The directed C-S 2-spanner problem is the problem finding the sparsest $S$ 2-spanner for $C$ on a directed graph. The C-S augmentation 2-spanner problem is a somewhat more general variant of the problem where it is assumed that a spanner $H$ for $C$ is already partially constructed, and the goal is to cheaply complete $H$ into a 2-spanner for $C$. More formally, consider the problem of finding the sparsest $S$ 2-spanner for $C$, where there is given a subset $L$ of $S$, which can be used without charge. The directed C-S 2-spanner augmentation problem is the C-S 2-spanner augmentation problem on a directed graph. We show that Algorithm A with minor changes provides $O(\log \frac{|C|}{|\tau(C)|})$ approximation ratio for all these three extensions of the C-S 2-spanner problem.

**Theorem 2** The directed C-S 2-spanner, the C-S 2-spanner augmentation and the directed C-S 2-spanner augmentation problems admit $O(\log \frac{|C|}{|\tau(C)|})$ approximation ratio.

The proof is omitted from this extended abstract.

6 **Bounded-Diameter Network Design Problems**

Let us first consider a simple special case of the BDND problem called the Unrestricted BDND (or UBDND) problem, in which $T = V \times V$.

Partition the requirements matrix $R$ (viewed as a graph over the $n$-vertex set $V$) into its connected components. For each $m$-vertex connected component $\tilde{R}$, the optimum network $\tilde{H}^*$ requires $m - 1$ edges, and this is achievable by constructing a star over $\tilde{R}$ with these many edges. It follows that the UBDND problem is polynomial.

The BDND problem also has an ordered variant, defined as follows. Suppose that a total order relation “<” is defined on $V$. Then in the ordered BDND problem, the path $(s_i, v_1), (v_1, v_2), \ldots, (v^{j-2}, v^{j-1}), (v^{j-1}, d_i)$ connecting $s_i$ and
\(d_i\) should satisfy also \(s_i < v^1 < v^2 < \ldots < v^{i-2} < v^{i-1} < d_i\). Thus the ordered UBDND problem is a special case of the directed BDND problem where \(T = \{(v, u) \in V \times V \mid v < u\}\). Note that the ordered UBDND problem is also polynomial, by the same argument as before.

We consider also another extreme version of the BDND problem in which we are allowed to use only edges from \(R\) to cover the requests, i.e., \(T = R\). We call this version the self-contained BDND (or SBDND) problem. In this case, we use the trivial reduction to the \(k\)-spanner problem, by setting \(E = R\). Therefore this version of 2-BDND problem admits an \(O(\log \frac{|V|}{n})\) approximation ratio for \(k = 2\) and an \(n^{O(1/k)}\) ratio for \(k > 2\) [12, 1]. Since there is a reduction in the opposite direction too, the hardness results of \(k\)-spanner problem are applicable here, i.e., the problem is NP-complete [12, 2], and it is NP-hard to approximate it for \(k > 2\) with better than \(O(\log n)\)-ratio [7].

The hardness results for the \(k\)-SBDND problem apply to the \(k\)-BDND problem as well. Also the same reduction to the C-S \(k\)-spanner problem (with \(S = T\), \(C = R\)) shows that the problem admits an \(O(\log \frac{|V|}{n})\) approximation ratio for \(k = 2\). Since there is a reduction in the opposite direction too, the hardness results concerning the C-S \(k\)-spanner problem apply to the BDND problem with diameter \(k\). In particular, it follows from our results in [3] that for every \(k > 2\), the problem is \(2^{O(\epsilon n)}\)-inapproximable for any \(0 < \epsilon < 1\) (henceforth, strongly inapproximable).

We also show in [3] that the C-S \(k\)-spanner problem, and thus the BDND problem with diameter \(k\), do not enjoy the ratio degradation property, i.e., in particular, cannot be approximated within a ratio of \(n^{O(1/k)}\).

We next show that the ordered \(k\)-BDND problem enjoys an approximation preserving reduction to the directed \(k\)-spanner problem, and therefore admits \(O(\log \frac{|V|}{n})\)-approximation ratio for \(k = 2\). For the directed \(k\)-spanner problem, like the C-S \(k\)-spanner problem, it is known [3] that it is strongly inapproximable and does not enjoy the ratio degradation property (i.e., its approximation threshold does not decrease when \(k\) tends to infinity). Thus all the hardness results mentioned above hold for this case too.

We devise the following reduction from the ordered BDND problem to the client-server directed \(k\)-spanner problem. Given an instance \((V, R, T)\) of the ordered \(k\)-BDND problem, we define an instance of the C-S directed \(k\)-spanner problem by setting

\[E = \{(v_i, v_j) \mid 1 \leq i < j \leq n\}, \quad C = R, \quad S = T.\]

**Lemma 6** A \(k\)-spanner \(H\) for the graph \(G\) is a feasible solution for the BDND problem \((V, R, T)\), and vice versa.

**Theorem 3** The (ordered) 2-BDND enjoys an \(O(\log \frac{|V|}{n})\) ratio approximation algorithm.
Next, let us consider the BDND augmentation problem. In this version of the BDND problem we are given also a subset $L$ of $V \times V$ of already “built” edges. We are interested in finding a minimum cardinality edge set $E$ disjoint of $L$, i.e., such that $E \subseteq T \setminus L$, and such that $E \cup L$ satisfies (1) (i.e., $E \cup L$ is a feasible solution for $R$). Using a similar reduction to the client-server $k$-spanner augmentation problem, we obtain an $O(\log \frac{|T|}{n})$ approximation algorithm for the 2-BDND augmentation problem that is applicable to all the above mentioned versions of the problem.

Let us now turn to discussing hardness of approximation. It is shown in [3] that the $k$-spanner augmentation problem is strongly inapproximable for $k > 3$. Since the SBDND problem is as hard as augmentation $k$-spanner problem, a similar result holds. With order restriction, the problem becomes as hard as the directed $k$-augmentation problem, and hence in this case the hardness result applies already for $k > 2$.

Consider the weighted SBDND problem. In this version of the BDND problem we are given also an $n \times n$ matrix $M$, where entry $M_{ij}$ represents the cost of building the edge between $v_i$ and $v_j$. This problem is reducible to the weighted client-server $k$-spanner problem, and therefore admits $O(\log n)$ approximation ratio for $k = 2$.

Again, the weighted $k$-SBDND problem is as hard as the weighted $k$-spanner problem, hence by [3]

**Theorem 4**

1. The weighted $k$-SBDND problem is strongly inapproximable for $k > 2$.

2. The 2-BDND problem in all the above mentioned versions (ordered, weighted, augmentation, self-contained) admits an $O(\log \frac{|T|}{n})$ approximation ratio algorithm.

3. $k$-SBDND problem is weakly ($O(\log n)$) inapproximable for $k \geq 2$.

4. The following problems are strongly inapproximable: BDND, ordered SBDND, weighted SBDND for $k > 2$, SBDND augmentation for $k > 3$.

**Remark:** All the hardness results for the SBDND problems apply also to the BDND variants; conversely, all the positive results for the BDND problems apply also to the SBDND variants.

**Acknowledgements**

We are grateful to one of the anonymous referees for helpful comments and remarks.
References


Michael Elkin is with the Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: elkin@wisdom.weizmann.ac.il

David Peleg is with the Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: peleg@wisdom.weizmann.ac.il
$k$-Neighborhood Broadcasting*

GUILLAUME FERTIN  
IRIN, Université de Nantes, France

ANDRÉ RASPAUD  
LaBRI, Université Bordeaux 1, France

Abstract

Broadcasting refers to the task whereby a node in a network, knowing a piece of information, must transmit it to all the other nodes. In this paper, we consider a generalized form of broadcasting, that we call $k$-neighborhood broadcasting. It consists in the following: a node $u$ in the network has to send its information to all the nodes which are at distance less than or equal to $k$ from $u$.

We study $k$-neighborhood broadcasting (or $k$-NB for short) in paths, trees, cycles, 2-dimensional grids and 2-dimensional tori under the store and forward, 1-port, unit cost model. For most of these families, we give the optimal $k$-NB time; if not, the optimal $k$-NB time is given within an additive constant never exceeding 2.

Keywords

Broadcasting, communications, networks.

1 Introduction and Preliminaries

We model a network by a graph, where the vertices and the edges represent respectively the processors and the communication links of the network. In such networks, broadcasting consists, for a source node $u$ of the network holding some information, in sending this information to all the other nodes in the network. Broadcasting has been widely studied in the previous years, and has given rise to several books and articles. We refer to [10, 5, 3, 11] for comprehensive surveys on this problem.

In this paper, we consider a generalized version of this problem, that we call $k$-neighborhood broadcasting (or $k$-NB for short); here, broadcasting must be realized at distance $k$ in the graph $G$, $1 \leq k \leq D(G)$, where $D(G)$ denotes the diameter of $G$. More formally, $k$-NB from $u$ consists in sending $u$’s information to

*Part of this research was done while the first author was visiting Simon Fraser University, Vancouver, Canada.
all the nodes which are within distance $k$ from $u$.

The model we use here is store-and-forward (messages go to their destination by transiting in the intermediate nodes; that is, communications always take place between neighbors), 1-port (at each given time, a node can communicate with at most one of its neighbors) and unit cost (a communication between two neighbors takes one time unit, or round).

When $k = D(G)$, then the $k$-NB problem is the usual broadcasting problem, and as mentioned earlier, a wide literature on the subject exists. We note that $k$-NB has also been studied in the particular case when $k = 1$ in two different families of graphs, namely hypercubes [2, 1] and star graphs [9].

We note that even in the case $k = 1$, the time to achieve 1-NB in a graph $G$ is not necessarily simply equal to the maximum degree $\Delta(G)$ of the graph. To see this, consider the wheel graph $W_{1,n}$ that consists of a cycle of order $n$, $C_n$, to which we add a “center” vertex $c$, connected to all the vertices of $C_n$. In that case, it can be seen that the time to achieve 1-NB from any vertex different from $c$ is equal to 2, and the time to achieve 1-NB from $c$ is equal to $\lceil 1+\sqrt{1+4(\Delta-1)} \rceil$, though the degree of $c$, $d(c)$, satisfies $d(c) = \Delta(W_{1,n}) = n$.

Let us also mention that another usual type of communication in networks, gossiping (where every node must send its information to every other node) can be similarly generalized to $k$-neighborhood gossiping (or $k$-NG). In the particular case $k = 1$, the 1-NG problem has been studied in hypercubes [6] and star graphs [9]; for general $k$, a study of $k$-NG, similar to the one presented in this paper, has been undertaken in [7].

In this paper, we give bounds on the optimal time for $k$-NB in the following families of graphs: paths, trees, cycles, 2-dimensional grids and tori. In most of those families, we determine the optimal $k$-NB time; if not, we give bounds on the optimal $k$-NB time, which differ by an additive constant not exceeding 2. In Section 2, we give some necessary definitions and determine some general properties about the minimum time to achieve $k$-NB in a graph $G$. In Sections 3 and 4, we study $k$-NB in paths, trees of maximum degree $\Delta$ and diameter $D$, cycles, 2-dimensional grids and 2-dimensional tori.

## 2 Generalities

We denote by $t_{NB}(G,u,k)$ the minimum time to achieve $k$-NB from node $u$ in the graph $G$ under our model. We call the $k$-neighborhood broadcast time of $G$, or $t_{NB}(G,k)$, the maximum $t_{NB}(G,u,k)$ over all $u \in V(G)$. Let also $N_k(G,u)$ be the number of vertices which are at distance less than or equal to $k$ from $u$ (u included), and $M_k(G) = \max_{u \in V(G)} N_k(G,u)$. We then have the following property.
Property 1 For any graph $G$ and for any $1 \leq k \leq D(G)$,
\[ t_{NB}(G,k) \geq \max\{k, \left\lceil \log_2(N_k(G)) \right\rceil \} \]

Proof. Any vertex $u$ in a graph $G$ which has to inform $t$ vertices (including $u$), needs at least $\left\lceil \log_2(t) \right\rceil$ time units in our model (at each time unit, the number of informed nodes can at most double). Since in $k$-NB, there exists at least one vertex $u$ in $G$ that must inform $N_k(G)$ vertices, we then get $t_{NB}(G,k) \geq \left\lceil \log_2(N_k(G)) \right\rceil$. Finally, it is easy to see that $t_{NB}(G,k) \geq k$ : indeed, $k \leq D(G)$, hence there always exist two vertices $u$ and $v$ at distance $k$. Since $u$ needs to broadcast its information to $v$, we have $t_{NB}(G,u,k) \geq k$, and thus $t_{NB}(G,k) \geq k$. \hfill $\Box$

Property 2 Let $G$ be a bipartite graph and $u$ a vertex of $G$. For any $1 \leq k \leq D(G)$ and any $p \geq 0$, we denote by $N(k,p)$ the maximum number of vertices at distance exactly $k$ from $u$ that can be informed by $u$ within $k + p$ rounds. In that case,
\[ N(k,p) \leq \frac{k + p}{2} \binom{k + p}{k + 2i} \left( \sum_{i=0}^{\left\lfloor \frac{k}{2} \right\rfloor} \right) . \]

Proof. Let $u \in V(G)$. If $u$ is to inform a vertex $v$ at distance exactly $k$ in $k + p$ rounds, it can either use a shortest path, or it can reach $v$ via a longer path. However, since $G$ is a bipartite graph, the latter only happens when the length of this path is of the same parity as $k$. Thus, in each case, a path of length $k + 2i$, $0 \leq i \leq \left\lfloor \frac{k}{2} \right\rfloor$, can be used to reach a vertex at distance $k$. In order to determine how many such paths are possible, it suffices to note that the number of broadcast sequences from $u$ to $v$ along a path of length $k + 2i$, using rounds between $1$ and $k + p$, is equal to the binomial $\binom{k + p}{k + 2i}$. Summing this term over all the possible values of $i$ (i.e., $0 \leq i \leq \left\lfloor \frac{k}{2} \right\rfloor$), we get the result. \hfill $\Box$

Thanks to the above property, and thanks to a well-known observation, we can deduce the following proposition, that we will use several times in the rest of this paper.

Proposition 1 For any graph $G$, let $u$ be a vertex of $G$. For any $1 \leq k \leq D(G)$, let $s_k(u)$ be the number of vertices at distance exactly $k$ from $u$. Then:
(a) If $s_k(u) \geq 2$, $t_{NB}(G,k) \geq k + 1$;
(b) If $G$ is bipartite and $s_k(u) \geq k + 2$, $t_{NB}(G,k) \geq k + 2$.

In the following, we denote by $G \square H$ the cartesian product of graphs $G$ and $H$.

Proposition 2 For any graphs $G$ and $H$, and for any $1 \leq k \leq \min\{D(G), D(H), D(G \square H)\}$, $t_{NB}(G \square H,k) \leq t_{NB}(G,k) + t_{NB}(H,k)$. 


Proof. The broadcast scheme we propose here from any vertex \((u, v) \in V(G \square H)\) is the following: achieve a \(k\)-NB from \((u, v)\) to all the vertices of the form \((u', v)\) that are in the copy of \(G\) containing \((u, v)\). Then, achieve a \(k\)-NB from any of the vertices \((u', v)\) reached by the first part of the scheme, to every vertex \((u', v')\) within distance \(k\) from \((u', v)\), that lies in the copy of \(H\) containing \((u', v)\). Clearly, this scheme respect the 1-port model, and is achieved in time \(t_{NB}(G, k) + t_{NB}(H, k)\). Moreover, this is a \(k\)-NB scheme: indeed, it can be easily seen that every vertex within distance \(k\) from \((u, v)\) is informed.

\[\]

3 Paths, Trees and Cycles

3.1 Paths

Theorem 1 Let \(P_n\) be the path of order \(n\). For any \(n\) and \(1 \leq k \leq n - 1\),

\[t_{NB}(P_n, k) = \begin{cases} 
    k & \text{if } k > \left\lfloor \frac{n-1}{2} \right\rfloor \\
    k + 1 & \text{otherwise}
\end{cases}
\]

Proof. By Property 1, we know that for any \(1 \leq k \leq D(P_n)\), that is for any \(1 \leq k \leq n - 1\), we have \(t_{NB}(P_n, k) \geq k\). Moreover, let us show that when \(1 \leq k \leq \left\lfloor \frac{n-1}{2} \right\rfloor\), there exists at least one vertex \(u\) that has to inform \(2\) vertices at distance \(k\), which implies by Proposition 1(a) that \(t_{NB}(P_n, u, k) \geq k + 1\). For this, let us denote by \(u_1, u_2, \ldots, u_n\) the vertices of \(P_n\), from left to right, and let us look at vertex \(u_{\left\lfloor \frac{n-1}{2} \right\rfloor + 1}\). It has \(\left\lfloor \frac{n-1}{2} \right\rfloor\) vertices on “its left” (that is, vertices denoted \(u_i\) with \(1 \leq i \leq \left\lfloor \frac{n-1}{2} \right\rfloor\)), and \(n - 1 - \left\lfloor \frac{n-1}{2} \right\rfloor = \left\lceil \frac{n-1}{2} \right\rceil \geq \left\lfloor \frac{n-1}{2} \right\rfloor\) vertices on “its right”. Hence, for any \(1 \leq k \leq \left\lfloor \frac{n-1}{2} \right\rfloor\), \(t_{NB}(P_n, k) \geq k + 1\). Note also that for any \(\left\lfloor \frac{n-1}{2} \right\rfloor < k \leq n - 1\), then for any \(u_i \in V(P_n)\), there is at most one vertex at distance \(k\) from \(u_i\).

Now, for any vertex \(u_i\) in \(P_n\) and every \(1 \leq k \leq n - 1\), the broadcast scheme is as follows: if \(u_i\) has two vertices at distance \(k\), then broadcast along the right (resp. left) edges from round 1 to round \(k\) (resp. from round 2 to round \(k + 1\)). If \(u_j\) is the only vertex at distance \(k\) from \(u_i\), then if \(j > i\) (resp. \(j < i\)) broadcast from round 1 to round \(k\) on the right (resp. left) edges, and from round 2 to round \(|j - i| + 1\) on the left (resp. right) ones. If there is no vertex at distance \(k\) from \(u_i\) then broadcast to the left from round 1, and to the right from round 2. In all the cases, it can be seen that the broadcast scheme we propose here meets the appropriate lower bound.

Non surprisingly, in the case \(k = D(P_n) = n - 1\), we obtain the same result as in [4] concerning the usual broadcast problem, that is \(t_{NB}(P_n, n - 1) = n - 1\).
3.2 Trees of maximum degree $\Delta$ and diameter $D$

In this section, we determine $t_{NB}$ for the family $\mathcal{T}_{\Delta,D}$ of trees with maximum degree $\Delta$ and diameter $D$. We denote by $t_{NB}(\mathcal{T}_{\Delta,D},k)$ the maximum $k$-NB time over all the trees $T \in \mathcal{T}_{\Delta,D}$.

**Definition 1 (\(\mathcal{T}_0(\Delta,D)\) and \(\mathcal{T}_1(\Delta,D)\))** For any fixed $\Delta$ and even $D$, we denote by $\mathcal{T}_0(\Delta,D)$ the unique tree rooted at $r$ such that the leaves lie at distance exactly $\frac{D}{2}$ from $r$, and such that all the vertices (except the leaves) are of degree $\Delta$.

For any odd $D = 2D' + 1$, $\mathcal{T}_1(\Delta,D)$ can be obtained from 2 copies $T_1$ and $T_2$ of $\mathcal{T}_0(\Delta,2D')$, where we delete a branch incident to the root $r_j$ in each of the $T_i$s ($i \in \{1,2\}$), and where we then connect the two roots $r_1$ and $r_2$ by an edge.

Examples of such trees are given in Figure 1, where we show respectively $\mathcal{T}_0(4,4)$ and $\mathcal{T}_1(4,5)$.

![Figure 1: $\mathcal{T}_0(4,4)$ (left) and $\mathcal{T}_1(4,5)$ (right).](image)

Clearly, $\mathcal{T}_0(\Delta,D) \in \mathcal{S}_{\Delta,D}$ (resp. $\mathcal{T}_1(\Delta,D) \in \mathcal{T}_{\Delta,D}$) for any fixed $\Delta$ and even $D$ (resp. odd $D$). More generally, for any fixed $\Delta$, it is easy to see that for any even $D$, any tree $T \in \mathcal{S}_{\Delta,D}$ is a subtree of $\mathcal{T}_0$, and for any odd $D$, any tree $T \in \mathcal{T}_{\Delta,D}$ is a subtree of $\mathcal{T}_1$.

We also note that for odd $D$, $\mathcal{T}_1(\Delta,D)$ can be seen as a copy of $\mathcal{T}_0(\Delta,D + 1)$ to which half of the leaves (in either the left or right subtree of $r$) has been removed.

**Lemma 1** For any fixed $\Delta$ and $D$, and for any $1 \leq k \leq \left\lfloor \frac{D}{2} \right\rfloor$:

(a) if $D$ is even, $t_{NB}(\mathcal{T}_0(\Delta,D),k) \geq k \cdot (\Delta - 1) + 1$;

(b) if $D$ is odd, $t_{NB}(\mathcal{T}_1(\Delta,D),k) \geq k \cdot (\Delta - 1) + 1$.

**Proof.** Suppose first $D$ is even, and let us show that for any $1 \leq k \leq \frac{D}{2}$, $t_{NB}(\mathcal{T}_0(\Delta,D),r,k) \geq k \cdot (\Delta - 1) + 1$, where $r$ is the root of $\mathcal{T}_0(\Delta,D)$. This is done by induction on $k$. When $k = 1$, it takes time $\Delta$ for $r$ to inform all vertices at distance $k$. Now suppose that $t_{NB}(\mathcal{T}_0(\Delta,D),r,k) \geq k \cdot (\Delta - 1) + 1$ for a fixed $k \leq \frac{D}{2}$, and let us show that this implies $t_{NB}(\mathcal{T}_0(\Delta,D),r,k+1) \geq (k+1) \cdot (\Delta - 1) + 1$, provided that $k + 1 \leq \frac{D}{2}$. There exists a vertex $v$ in $\mathcal{T}_0(\Delta,D)$ at distance $k$ from $r$, which is informed after round $k \cdot (\Delta - 1) + 1$. Moreover, since $k + 1 \leq \frac{D}{2}$, $v$ has $\Delta - 1$ sons in $\mathcal{T}_0(\Delta,D)$ which are at distance $k + 1$ from $r$. Hence it will take at
least $\Delta - 1$ more rounds for $v$ to inform its $\Delta - 1$ sons in $T_0(\Delta, D)$, and thus we have $t_{NB}(T_0(\Delta, D), r, k+1) \geq k \cdot (\Delta - 1) + 1 + \Delta - 1$. Thus the result is proved by induction on $k$.

Now suppose $D$ is odd. In that case, for any $k \leq \frac{D-1}{2}$, it is easy to see that the subtree of $T_1(\Delta, D)$ induced by the vertices at distance less than or equal to $k$ from $r_1$ is isomorphic to $T_0(\Delta, 2k)$, with $r = r_1$. Thus, since by Lemma 1(a), we know that $t_{NB}(T_0(\Delta, 2k), r, k) \geq k \cdot (\Delta - 1) + 1$, we get $t_{NB}(T_1(\Delta, D), r, k) \geq k \cdot (\Delta - 1) + 1$. Hence $t_{NB}(T_1(\Delta, D), k) \geq k \cdot (\Delta - 1) + 1$, and the result is proved. \hfill $\square$

**Lemma 2** For any fixed $\Delta, D$ and $1 \leq k \leq \lfloor \frac{D}{2} \rfloor$:

(a) if $D$ is even, $t_{NB}(T_0(\Delta, D), k) \leq k \cdot (\Delta - 1) + 1$;

(b) if $D$ is odd, $t_{NB}(T_1(\Delta, D), k) \leq k \cdot (\Delta - 1) + 1$.

**Proof.** First suppose $D$ is even. Let $v$ be any vertex of $T_0(\Delta, D)$. The main idea here is to note that the subtree of $T_0(\Delta, D)$ induced by the vertices that are at distance less than or equal to $k$ from $v$ is itself a subtree $T$ of $T_v = T_0(\Delta, 2k)$, where $T_v$ is rooted at $v$ (cf. Figure 2, where $\Delta = 4, D = 6$ and $k = 3$). However, in $T_v$, and consequently in $T$, we know that we can achieve $k$-NB from $v$ within $k \cdot (\Delta - 1) + 1$ rounds. Hence, for any vertex $v \in T_0(\Delta, D)$, we have $t_{NB}(T_v(\Delta, D), v, k) \leq k \cdot (\Delta - 1) + 1$. This proves the result when $D$ is even.

Similarly, when $D$ is odd, for any $v \in T_1(\Delta, D)$, the subtree $T'_v$ induced by the set of vertices that lie within distance $k$ of $v$ in $T_1(\Delta, D)$ is itself a subtree of $T'_v = T_0(\Delta, 2k)$, where $T'_v$ is rooted at $v$. Since we know that in $T'_v$ (and thus in $T'$) it is possible to achieve $k$-NB from $v$ within $k \cdot (\Delta - 1) + 1$ rounds, we then have $t_{NB}(T'_v(\Delta, D), v, k) \leq k \cdot (\Delta - 1) + 1$ for any $v$ in $T_1(\Delta, D)$, and thus the result is also proved for odd $D$. \hfill $\square$

**Lemma 3** For any fixed $\Delta, D$ and $\lfloor \frac{D}{2} \rfloor + 1 \leq k \leq D$:

(a) if $D$ is even, $t_{NB}(T_0(\Delta, D), k) \geq \frac{D}{2} \cdot (\Delta - 2) + k$;

Figure 2: In $T_0(4, 6)$, we construct $T$ rooted at $v$, with $k = 3$. 

- [Diagram of Figure 2]
In other words, any $k$ rounds. Since \( r \) and \( v \) must be informed before round \( r \) in \( T_0(\Delta, D) \) (such a vertex exists, since \( k \leq D \)). Actually, \( v \) is chosen such that it must broadcast to all the other vertices in the tree. Let \( r_v \) be the son of \( r \) that is on the path from \( v \) to \( r \), and let \( T' \) be the subtree of \( T_0(\Delta, D) \) rooted at \( r_v \), and obtained by deleting the edge \( rr_v \). Any vertex not in \( T' \) must be informed via \( r_v \); that is, \( r \) must inform all the vertices of \( T_r = T_0(\Delta, D) - T' \). Moreover, \( r \) cannot be informed before round \( R_1 = k - \frac{D}{2} \). Now, in order for \( r \) to inform all the vertices of \( T_r \), it can be seen that we need at least \( R_2 = \frac{D}{2}(\Delta - 1) \) rounds. Summing \( R_1 \) and \( R_2 \), we get \( t_{NB}(T_0(\Delta, D), v, k) \geq \frac{D}{2}(\Delta - 2) + k \).

Now let us assume \( D \) is odd. In that case, we use the same kind of argument: let \( T_1 \) (resp. \( T_2 \)) be the subtree of \( T_1(\Delta, D) \) that contains \( r_1 \) (resp. \( r_2 \)) that is obtained by removing the edge \( r_1r_2 \), and let \( v \) be a vertex lying at distance \( \frac{D}{2} \) from \( r_1 \) in \( T_1 \). \( v \) has been chosen so that it must broadcast to all the vertices in \( T_1(\Delta, D) \). In particular all the vertices of \( T_2 \) must be informed via \( r_1 \) and \( r_2 \). Moreover, if we denote by \( r_1' \) the son of \( r_1 \) that lies on the path from \( v \) to \( r_1 \), and if we denote by \( T' \) the subtree containing \( r_1' \) obtained by deleting the edge \( r_1'r_2 \), all the vertices of \( T_1 - T' \) must also be informed via \( r_1 \). Whatever the strategy used, and in particular whatever round is used on edge \( r_1r_2 \), it can be shown that there exists a leaf in \( T' \) that cannot be informed by \( u \) in less than \( k - \frac{D}{2} + (\Delta - 1) \) \( \frac{D}{2} \) rounds. In other words, we have \( t_{NB}(T_1(\Delta, D), v, k) \geq \frac{D}{2}(\Delta - 2) + k \) and thus the result is proved.

\[ \square \]

**Lemma 4** For any fixed \( \Delta, D \) and \( \lfloor \frac{D}{2} \rfloor + 1 \leq k \leq D \):

(a) if \( D \) is even, \( t_{NB}(T_0(\Delta, D), k) \leq \frac{D}{2}(\Delta - 2) + k \);

(b) if \( D \) is odd, \( t_{NB}(T_1(\Delta, D), k) \leq \lfloor \frac{D}{2} \rfloor - (\Delta - 2) + k \).

**Proof.** Suppose first \( D \) is even, and let \( \frac{D}{2} + 1 \leq k \leq D \). First, we know that every vertex in \( T_0(\Delta, D) \) is at distance at most \( \frac{D}{2} \) from \( r \). Hence, any \( k \)-NB from \( r \) with \( k \geq \frac{D}{2} + 1 \) is equivalent to a \( \frac{D}{2} \)-NB from \( r \), and thus takes at most \( \frac{D}{2}(\Delta - 1) + 1 \) rounds. Since \( k \geq \frac{D}{2} + 1 \), we see that we always have \( \frac{D}{2}(\Delta - 1) + 1 \leq k + \frac{D}{2}(\Delta - 2) \). In other words, any \( k \)-NB from \( r \) always satisfies the inequality (a) of Lemma 4 above.

Now, for any vertex \( v \neq r \) in \( T_0(\Delta, D) \), let \( dist(v, r) \) be its distance from the root \( r \), and let the local root \( lr \) be defined as follows: \( lr = r \) if \( dist(v, r) \leq k - \frac{D}{2} \), and \( lr \) is the vertex lying at distance \( k - \frac{D}{2} \) from \( v \) on the path from \( v \) to \( r \) otherwise. Two cases then arise:

(a) \( lr = r \). In that case, this means that \( v \) must broadcast to all the vertices
in the tree. Let \( v \) inform \( r \) using rounds \( 1, 2, \ldots, \text{dist}(v, r) \). Let us denote by \( r' \) the son of \( r \) that lies on the path from \( v \) to \( r \), and by \( T' \) the tree rooted at \( r' \) that is obtained by removing the edge \( rr' \). Now, \( r \) informs all the vertices of \( T_0(\Delta, D) - T' \) starting from round \( \text{dist}(v, r) + 1 \). This is achieved using at most \( \frac{D}{2}(\Delta - 2) + k \) rounds on the whole. During that time, \( r' \) broadcasts in \( T' \) (of depth \( \frac{D}{2} - 1 \)), with the constraint that the path from \( v \) to \( r \) is already allowed some rounds. However, this can be achieved within round \( \frac{D}{2}(\Delta - 2) + k - \Delta \). Thus, globally, the broadcast scheme uses at most \( \frac{D}{2}(\Delta - 2) + k \) rounds.

(b) \( lr \neq r \). In that case, \( v \) does not need to broadcast to all the vertices in the tree: its \( k \)-neighborhood does not contain every vertex in the tree. However, using the same notation as previously, it must broadcast to all the vertices in \( T' \), and to all the vertices lying at distance less than or equal to \( \frac{D}{2} \) from \( lr \) in the tree \( T_0(\Delta, D) - T' \). One can also see that the depth of \( T' \) is less than or equal to \( \frac{D}{2} \); hence, globally the broadcast scheme described in the previous case remains the same, except that \( lr \) is substituted to \( r \). The maximum number of rounds needed here is also \( \frac{D}{2}(\Delta - 2) + k \), and thus the result is proved.

When \( D \) is odd, we use a very similar kind of argument to show that in \( T_1(\Delta, D), k \)-NB can be achieved from any vertex \( v \) in \( \frac{D}{2} \cdot (\Delta - 2) + k \) rounds. \( \Box \)

**Theorem 2** \( t_{NB}(T_{\Delta, D}, k) = \begin{cases} \frac{k \cdot (\Delta - 1) + 1}{2} & \text{if } 1 \leq k \leq \frac{D}{2} \\ \frac{k \cdot (\Delta - 2) + k}{2} & \text{if } \frac{D}{2} < k \leq D \end{cases} \)

**Proof.** We know that any tree \( T \in T_{\Delta, D} \) is a subtree of \( T_0(\Delta, D) \) when \( D \) is even (resp. a subtree of \( T_1(\Delta, D) \) when \( D \) is odd). Thus we have \( t_{NB}(T, k) \leq t_{NB}(T_0(\Delta, D), k) \) for any \( T \in T_{\Delta, D} \) with even \( D \), and \( t_{NB}(T, k) \leq t_{NB}(T_1(\Delta, D), k) \) for any \( T \in T_{\Delta, D} \) with odd \( D \).

Moreover, by Lemmas 1 to 4 we have:

- If \( D \) is even, then \( t_{NB}(T_0(\Delta, D), k) = \begin{cases} k \cdot (\Delta - 1) + 1 & \text{when } 1 \leq k \leq \frac{D}{2} \\ \frac{D}{2} \cdot (\Delta - 2) + k & \text{when } \frac{D}{2} < k \leq D \end{cases} \)

- If \( D \) is odd, then

\( t_{NB}(T_1(\Delta, D), k) = \begin{cases} k \cdot (\Delta - 1) + 1 & \text{when } 1 \leq k \leq \frac{D - 1}{2} \\ \frac{D - 1}{2} \cdot (\Delta - 2) + k & \text{when } \frac{D - 1}{2} < k \leq D \end{cases} \)

Hence the result. \( \Box \)

**Remark:** We note that Theorem 1 is a particular case of Theorem 2 above. Indeed, if we suppose \( \Delta = 2 \), \( T_{\Delta, D} \) only contains one tree, more precisely the path \( P_{D+1} \). Putting now \( n = D + 1 \) gives directly Theorem 1.
We also note that there exists several results concerning the (usual) broadcast time in some specific types of trees, such as complete \( \Delta \)-ary trees of depth \( h \) (cf. for instance [3]). However, these results cannot be compared to ours, since in complete \( \Delta \)-ary trees, every vertex not being a leaf has \( \Delta \) sons. This means that in complete \( \Delta \)-ary trees, the root is of degree \( \Delta \), and every other vertex not being a leaf is of degree \( \Delta + 1 \); in our case, every vertex not a leaf (root included) is of degree \( \Delta \). This is not to mention that complete \( \Delta \)-ary trees of depth \( h \) are of even diameter \( D = 2h \), and that in our case we do not restrict \( D \) to be even.

**3.3 Cycles**

**Theorem 3** Let \( C_n \) be the cycle of order \( n \). For any \( n \geq 3 \) and any \( 1 \leq k \leq \left[ \frac{n}{2} \right] \):

\[
t_{NB}(C_n, k) = \begin{cases} \\
  k & \text{if } n \text{ is even and } k = \frac{n}{2} \\
  k + 1 & \text{otherwise}
\end{cases}
\]

**Proof.** For any \( n \), \( C_n \) is vertex-transitive, thus we can focus on broadcasting from a given vertex \( v \in V(C_n) \). Suppose first \( n \) is odd. In that case, for any \( 1 \leq k \leq \left[ \frac{n}{2} \right] \), there are two vertices at distance \( k \) from \( v \), and thus by Proposition 1(a) \( t_{NB}(C_n, k) \geq k + 1 \). When \( n \) is even, there are also 2 vertices at distance \( k \) from \( v \), except in the case \( k = \frac{n}{2} \), where there is only one such vertex. Thus, when \( n \) is even, \( t_{NB}(C_n, k) \geq k + 1 \) when \( k \neq \frac{n}{2} \) by Proposition 1(a) and \( t_{NB}(C_n, k) \geq k \) otherwise.

Now let us show that these bounds are tight. For this, let \( v \) broadcast its information at distance \( k \) in \( C_n \) as follows: at round 1, \( v \) sends its information to one of its neighbor, and in the following rounds, every newly informed vertex sends the information to its still uninformed neighbor. Following this scheme, it can be seen that any \( k \)-NB in \( C_n \) is achieved in \( k + 1 \) rounds, except in the case \( n \) even and \( k = \frac{n}{2} \), where \( k \) rounds suffice (since, as mentioned above, for any \( v \in V(C_n) \) there exists only one vertex \( v' \) at distance \( k \) from \( v \)).

Here again, we note that in the case \( k = D(C_n) = \left[ \frac{n}{2} \right] \), we obtain the well known result concerning (usual) broadcasting in cycles; that is, \( t_{NB}(C_n, \left[ \frac{n}{2} \right]) = \left[ \frac{n}{2} \right] \).

**4 2-Dimensional Grids and Tori**

In the following, \( G(n_1, n_2) \) (resp. \( TG(n_1, n_2) \)) will denote the 2-dimensional grid (resp. torus) with \( n_1 \) rows and \( n_2 \) columns. Due to symmetries, we will always suppose \( n_1 \geq n_2 \). We also denote here by \( D(G(n_1, n_2)) \) the diameter of the 2-dimensional grid \( G(n_1, n_2) \), and by \( D(TG(n_1, n_2)) \) the diameter of the 2-dimensional torus \( TG(n_1, n_2) \). Thus we have \( D(G(n_1, n_2)) = n_1 + n_2 - 2 \) and
$$D(TG(n_1, n_2)) = \left\lfloor \frac{n_1}{2} \right\rfloor + \left\lfloor \frac{n_2}{2} \right\rfloor.$$  

### 4.1 2-Dimensional Grids

**Proposition 3 (1-NB in 2-dim. Grids)** For any integers \( n_1, n_2 \) with \( n_1 \geq n_2 \geq 2 \),

\[
\tau_{NB}(G(n_1, n_2), 1) = \begin{cases} 
2 & \text{if } n_1 = n_2 = 2 \\
3 & \text{otherwise}
\end{cases}
\]

**Proof.** When \( n_1 = n_2 = 2 \), it is easy to see that 2 rounds are necessary and sufficient for any vertex to broadcast at distance 1. In all the other cases, there exists in the grid at least one vertex \( v \) for which the number of vertices at distance 1 from \( v \) is greater than or equal to 3. Since \( G(n_1, n_2) \) is a bipartite graph, thanks to Proposition 1(b) we conclude that \( \tau_{NB}(G(n_1, n_2), 1) \geq 3 \). However, it is easy to see that 3 rounds are sufficient in that case (cf. for this Figure 3(left)). \( \square \)

**Proposition 4 (2-NB in 2-dim. Grids)** For any integers \( n_1, n_2 \) with \( n_1 \geq n_2 \geq 2 \),

\[
\tau_{NB}(G(n_1, n_2), 2) = \begin{cases} 
2 & \text{if } n_1 = n_2 = 2 \\
3 & \text{if } n_1 \in \{3,4\} \text{ and } n_2 = 2 \\
4 & \text{if } n_1 \geq 5 \text{ and } n_2 = 2 \text{ or } n_2 \in \{3,4\} \\
5 & \text{otherwise (that is, if } n_1 \geq n_2 \geq 5) 
\end{cases}
\]

**Sketch of Proof:** The proof relies here on detailing all the possible cases, noting that, for fixed \( n_2 \), for any \( n_1 \geq 5 \), the result is the same as for \( n_1 = 5 = 2k + 1 \). For any \( n_2 \leq 4 \), by a rather tedious case by case analysis, the above results can be deduced. If \( n_1 \geq n_2 \geq 5 \), we note that there exists a vertex \( v \) in \( G(n_1, n_2) \) that must inform \( 4k = 8 \) vertices at distance \( k = 2 \). By Property 2, if we are allowed \( k + 2 = 4 \) rounds, then \( v \) can inform at most \( \frac{(k+2)(k+1)}{2} + 1 = 7 \) vertices at distance 2. Thus, in that case \( \tau_{NB}(G(n_1, n_2), 2) \geq 5 \); it is easy to see that 5 rounds suffice for any vertex in \( G(n_1, n_2) \) (with \( n_1 \geq n_2 \geq 5 \)) to broadcast its information within distance 2 (cf. for instance Figure 3(middle)). \( \square \)

**Proposition 5 (3-NB in 2-dim. Grids)** For any integers \( n_1 \geq 3 \) and \( n_2 \geq 2 \) s.t. \( n_1 \geq n_2 \),

\[
\tau_{NB}(G(n_1, n_2), 3) = \begin{cases} 
3 & \text{if } n_1 = 3 \text{ and } n_2 = 2 \\
4 & \text{if } n_1 \in \{4,5,6\} \text{ and } n_2 = 2 \\
\text{or if } n_1 \in \{3,4\} \text{ and } n_2 = 3 \text{ or } n_1 = n_2 = 4 \\
5 & \text{if } n_1 \geq 7 \text{ and } n_2 = 2 \text{ or if } n_1 \geq 5 \text{ and } n_2 \in \{3,4\} \\
\text{or if } n_1 \in \{5,6\} \text{ and } n_2 = 5 \\
6 & \text{otherwise}
\end{cases}
\]

**Sketch of Proof:** Here again, we detail all the possible cases up to \( G(7, 7) \). Moreover, when \( n_1 \geq n_2 \geq 7 \), by Property 2 we can show that \( k + 3 = 6 \) rounds are necessary to achieve a \( k \)-NB from any of the vertices of \( G(n_1, n_2) \); as in the previous

...
case, it is easy to show that 6 rounds are enough (cf. for instance Figure 3(right)).

Proposition 6 (k-NB in 2-dim. Grids) For any integers \( n_1, n_2 \) with \( n_1 \geq n_2 \geq 2 \) and \( n_1 + n_2 \geq 6 \), let \( \text{eq}(n_1,n_2) = 1 \) if \( n_1 = n_2 \) and 0 otherwise. Let also \( t_1 = \left\lfloor \frac{2n_1+n_2}{3} \right\rfloor - 1 - \text{eq}(n_1,n_2) \). We have:

(a) \( k + 3 \geq t_{NB}(G(n_1,n_2),k) \geq \begin{cases} k + 2 & \text{if } 4 \leq k \leq t_1 \\ k + 1 & \text{if } t_1 + 1 \leq k \leq D(G(n_1,n_2)) - 1 \end{cases} \)

(b) \( t_{NB}(G(n_1,n_2),k) = k \) if \( k = D(G(n_1,n_2)) \)

Proof. (b) is a direct application of the result of [4] concerning the usual broadcasting in 2-dimensional grids.

Now let us prove (a), and let us first prove the upper bound. For this, we give a \( k \)-NB achievable in \( k + 3 \) rounds from any vertex \( v \) in \( G(n_1,n_2) \). We will first suppose that \( n_1 \) and \( n_2 \) are “big enough”, that is we do not take the boundaries into account. Let us prove by induction that it is possible to achieve a \( k \)-NB in \( k + 3 \) rounds, with the property that the leftmost (resp. rightmost) vertex from the source, \( u_k \) (resp. \( v_k \)), is informed at round \( k + 1 \) at worse. Clearly, this property is satisfied for \( k = 2 \) and \( k = 3 \) (cf. Figure 3(middle) and (right)). Now, suppose this is true for a given \( k \geq 2 \), and let us show it still holds for \( k + 1 \). For this, let any vertex \( v \), different from \( u_k \) and \( v_k \), lying at distance \( k \) from the source, broadcast the information (that it obtained at round \( k + 3 \) at worse, by hypothesis) vertically during round \( k + 4 \). The vertices that are “above” the source broadcast upwards, the others downwards. Then 6 vertices remain uninformed : 3 are neighbors of \( u_k \) and \( v_k \), and 3 are neighbors of \( v_k \). Note that \( u_k \) and \( v_k \) are informed at round \( k + 1 \) at worse by induction hypothesis. Let \( u_k \) (resp. \( v_k \)) inform \( u_{k+1} \) on its left (resp. \( v_{k+1} \) on its right) during round \( k + 2 \), and inform its two still uninformed neighbors during rounds \( k + 3 \) and \( k + 4 \). This scheme is a \( (k + 1) \)-NB scheme, running in \( k + 4 \) rounds, with the added property that \( u_{k+1} \) (resp. \( v_{k+1} \)) is informed at round \( k + 2 \).
at worse. Hence the upper bound is proved by induction on \( k \).

Now, depending on the values of \( n_1 \) and \( n_2 \), the boundaries of the grid can sometimes interfere; however, it suffices to “cut” the figure where the boundaries appear, and one can see that it does not affect the scheme in the sense that all the vertices within distance \( k \) are still reached. Thus \( t_{NB}(G(n_1, n_2), k) \leq k + 3 \), for any \( k \geq 2 \) and any \( n_1 \geq n_2 \geq 2 \).

The lower bounds are obtained thanks to Proposition 1, \( G(n_1, n_2) \) being bipartite: when \( k \leq t_1 \), it is possible to find in \( G(n_1, n_2) \) a vertex \( v \) that has \( s_k(v) \geq k + 2 \) vertices at distance \( k \), thus \( t_{NB}(G(n_1, n_2), k) \geq k + 2 \). When \( t_1 + 1 \leq k \leq D(G(n_1, n_2)) - 1 \), there exists at least one vertex \( v \) that has \( s_k(v) \geq 2 \) vertices at distance \( k \), and thus \( t_{NB}(G(n_1, n_2), k) \geq k + 1 \). \( \square \)

### 4.2 2-Dimensional Tori

**Proposition 7** For any integers \( n_1 \) and \( n_2 \) with \( n_1 \geq n_2 \geq 3 \), we have:

1. \( t_{NB}(TG(n_1, n_2), 1) = 3 \)
2. \( k + 1 \leq t_{NB}(TG(n_1, n_2), k) \leq k + 3 \) for any \( 2 \leq k \leq D(TG(n_1, n_2)) - 1 \)
3. For any \( k = D(TG(n_1, n_2)) \),
   \[
   t_{NB}(TG(n_1, n_2), k) = \begin{cases} 
   k & \text{if } n_1 \text{ and } n_2 \text{ are even} \\
   k + 1 & \text{otherwise}
   \end{cases}
   \]

**Proof.** In the case \( k = 1 \), at least 3 rounds are necessary by Property 1, since there are 5 vertices to inform (source included). One can notice that the 1-NB scheme in 3 rounds given in Figure 3(left) also applies for tori: \( TG(n_1, n_2) \) being vertex-transitive, we conclude that \( t_{NB}(TG(n_1, n_2), k) \leq 3 \) and the equality holds.

In the case \( 2 \leq k \leq D(TG(n_1, n_2)) - 1 \), we note that the upper bound of \( k + 3 \) given in Proposition 6 for grids also holds in tori. Indeed, similarly as for the boundaries in grids, the wrap-around in tori does not interfere in the ability for a vertex to broadcast at distance \( k \). The lower bound of \( k + 1 \) in (b) derives from the fact that for every vertex \( u \in V(TG(n_1, n_2)) \), there exists at least two vertices lying at distance exactly \( k \) from \( u \).

Finally, the result (c) comes from [4], where the usual broadcasting in 2-dimensional tori has been studied. \( \square \)

**Proposition 8** For any integers \( n_1 \geq n_2 \geq 4 \) where \( n_1 \) and \( n_2 \) are both even, let \( t_2 = \left\lfloor \frac{2(n_1 + n_2 - 1)}{3} \right\rfloor \). In that case, we have \( t_{NB}(TG(n_1, n_2), k) \geq k + 2 \) in the following cases:

1. \( n_1 \geq 4n_2 - 2 \) and \( k \leq 2n_2 - 2 \)
2. \( n_1 = 4n_2 - 4 \) and \( k \leq 2n_2 - 3 \)
(c) \( n_1 \leq 4n_2 - 6 \) and \( k \leq t_2 \).

**Sketch of proof:** We note that when \( n_1 \) and \( n_2 \) are both even, then \( TG(n_1, n_2) \) is bipartite, and thus Proposition 1(b) applies in that case. Hence it suffices to detail the cases for which there exists in \( TG(n_1, n_2) \) at least \( k + 2 \) vertices at distance exactly \( k \) from a given vertex \( u \). The results given above describe exactly those cases. By Proposition 1(b), we conclude that in those cases, \( t_{NB}(TG(n_1, n_2), k) \geq k + 2 \).

\[ \square \]

5 Conclusion

In this paper, we have studied for the first time the \( k \)-NB for general \( k \) in the store-and-forward, 1-port, unit cost model. We have developed several methods and techniques that we have applied to several families of graphs, in order to obtain, as often as possible, exact results. In some other cases, we have given bounds that are close to the optimal (up to an additive constant never exceeding 2). Due to the length of the paper, we have not been able to state all our results. Notably, we are also able to determine good bounds for \( k \)-NB in \( d \)-dimensional grids and tori, triangular meshes and hypercubes. Also, as mentioned earlier, a similar study has been undertaken in the case of \( k \)-neighborhood gossiping [7]. All these results will be detailed in the journal version of this paper.

The main drawback that we wish to state here is the lack of generality of the methods we propose. We note that the same occurs in previous papers studying either 1-NB or 1-NG in specific families of graphs [2, 1, 6, 9]. Non surprisingly, some difficulties also occur when the studied topology is not vertex-transitive.

6 Acknowledgements

We wish to thank Joseph G. Peters for introducing us to the problem.

References


Guillaume Fertin is with IRIN, Université de Nantes - 2 rue de la Houssinière, BP 92208, 44322 Nantes Cedex 3 - France. E-mail: fertin@irin.univ-nantes.fr

André Raspaud is with LaBRI, Université Bordeaux 1 - 351 cours de la Libération, 33405 Talence Cedex - France. E-mail: raspaud@labri.fr
Pattern Formation by Autonomous Robots without Chirality

PAOLA FLOCCHINI  
*University of Ottawa, Canada*

GIUSEPPE PRENCIPE  
*Università di Pisa, Italy*

NICOLA SANTORO  
*Carleton University, Canada*

PETER WIDMAYER  
*ETH Zürich, Switzerland*

Abstract

Consider a set of anonymous mobile robots, and the patterns they can collectively form in the plane. If each robot has a compass needle that indicates *North* but there is no agreement on East and West (i.e., no chirality), it is known that every arbitrary pattern given in input can be formed if and only if the number of robots is odd. In this paper we study what patterns can be formed by an even number of such robots, and provide a complete characterization. We identify the class of patterns which cannot be formed by an even number of anonymous robots without chirality, regardless of additional capabilities. We then construct a set of rules (distributed algorithm) which leads the robots to form any of the other patterns in finite time. The algorithm is collision-free, completely oblivious and fully asynchronous.

Keywords

Mobile computing, robots, distributed algorithms, oblivious computations, chirality.

1 Introduction

In this paper we consider a collection of identical *anonymous mobile robots*, and the patterns they can collectively form in the plane.

Following the current trend in robotics research, our robots are anonymous, have minimal capabilities, and exhibit an extremely simple behavior. Their actions consist in autonomously observing the environment, computing a destination point, and moving towards it. Their behavior is a simple continuous cycle of sensing, computing, moving and being inactive.
The ability to form geometric patterns is a fundamental one; it is usually an initial step which allows the robots to decide on their respective roles in a subsequent, coordinated action (e.g., forming a circle around an object which has to be removed, arranging themselves in a line so to sweep the terrain in formation, etc.).

Pattern formation by a collection of such robots has been studied by researchers in robotics and in artificial intelligence (e.g., see [1, 3, 4]). Concerns on computability and complexity of the problem have recently motivated also algorithmic investigations [2, 6, 7, 8]; the basic difference is that in [7, 8], any action by the robots, including moving, is instantaneous. In this paper, like in [2, 6], there is no such an assumption.

The capacity to solve the pattern formation problem seems to depend not only to the robots’ (computing and motorial) capabilities but also and especially on the level of agreement they have on the coordinate system.

In fact, if the robots have complete agreement on a common coordinate system¹ (e.g., each robot has a compass), they can form any arbitrary pattern given in input; they can do so even if they are totally asynchronous in their actions (and inactions), and are completely oblivious (i.e., they do not remember any previous observation, computation, or action) [2]. On the other hand, if the robots have no agreement on the coordinate system, then the arbitrary pattern formation is unsolvable regardless of their capabilities (unbounded memory, instantaneous actions [2].

More interesting is the case when there is a limited agreement among the robots; that is, when they agree on the direction and orientation of one axis, but do not have a common understanding of the handedness (chirality) of the coordinate system²; thus knowing North does not distinguish East from West. In this case, an odd number of anonymous asynchronous oblivious robots can still form any arbitrary pattern given in input; on the other hand, if the number of robots is even, the arbitrary pattern formation problem is unsolvable regardless of their capabilities [2].

Since, with limited agreement, an even number of anonymous robots cannot form every pattern, the natural immediate questions are what patterns, if any, can they form?, what capabilities they need to do so?, how can they do it? In this paper we provide a complete constructive answer to these questions.

In fact, we first identify a class of patterns \( \Omega \) that, with partial agreement, cannot be formed by an even number of anonymous robots, regardless of their capabilities. The impossibility result holds even allowing the robots to remember the past and to perform synchronous and instantaneous computations. We then present a set of rules (distributed algorithm) which, when executed (asyn-

---

¹i.e., on the direction and orientation of the two axes but not necessarily on their origin nor on a common unit length.

²chirality allows to consistently infer the orientation of the \( Y \) axis once the orientation of the \( X \) axis is given.
chronously and independently) by the robots leads to form any input pattern $\mathbb{P} \notin \Omega$ in finite time. The pattern formation algorithm is \textit{completely oblivious} (i.e., it does not require any knowledge by the robots of past computations or observations), \textit{fully asynchronous} (i.e., it does not make any assumption on the speed or computation time of the robots), and \textit{collision-free} (i.e., two robots will never occupy the same position in the plane).

\section{Model and Basic Properties}

\subsection{The Robots}

We consider a system of autonomous mobile robots. Each robot is capable of sensing its immediate surrounding, performing computations on the sensed data, and moving towards the computed destination; its behavior is an (endless) cycle of sensing, computing, moving and being inactive.

The robots are viewed as points, and modeled as units with computational capabilities, which are able to freely move in the plane. They are equipped with sensors that let each robot observe the positions of the others and form its \textit{local view} of the world. This view includes a unit of length, an origin (which we will assume w.l.g. to be the position of the robot in its current observation), and a Cartesian coordinate system (comprising the directions of the two coordinate axes, referred to as $X$ and $Y$, and their orientations). We assume the robots agree on the two axes direction, but on the orientation of only one of them (w.l.g, let it be axis $Y$). Notice that such an agreement does not imply agreement on the unit distance nor on the location of the origin.

The robots are \textit{oblivious}, meaning that they do not need to remember any previous observation nor computations performed in the previous steps.

The robots are \textit{anonymous}, meaning that they are a priori indistinguishable by their appearances, and they do not need to have any kind of identifiers that can be used during the computation. Moreover, there are no explicit direct means of communication.

The robots are \textit{fully asynchronous}: there is no common notion of time, and the amount of time spent in observation\(^3\), in computation, in movement, and in inaction is finite but otherwise unpredictable.

The robots execute the same deterministic algorithm, which takes as input the observed positions of the robots within the visibility radius, and returns a destination point towards which the executing robot moves.

A robot is initially in a \textit{waiting} state (\textit{Wait}). Asynchronously and independently from the other robots, it \textit{observes} the environment (\textit{Look}) by activating its sensors which will return a snapshot\(^4\) of the positions of all other robots with

\footnote{i.e., activating the sensors and receiving their data.}

\footnote{We do not require the robots to be able to detect \textit{multiplicity} (i.e. whether there is more than one}
respect to its local coordinate system. (Since robots are viewed as points, their positions in the plane is just the set of their coordinates). The robot then calculates its destination point (Compute) according to its deterministic, oblivious algorithm, based only on the observed locations of the robots. It then moves towards that point (Move): if the destination point is the current location, the robot stays still (null movement). After the move, the robot becomes waiting again. The sequence Wait - Look - Compute - Move forms a cycle of a robot.

There are two limiting assumptions excluding the infinite: 1. The amount of time required by a robot to complete a cycle is not infinite, nor infinitesimally small. 2. The distance traveled by a robot in a cycle is not infinite, nor infinitesimally small (unless it brings the robot to the destination point).

As no other assumptions on time exists, the resulting system is truly asynchronous and the duration of each activity (or inactivity) is unpredictable. As a result, robots can be seen while moving, and computations can be made based on obsolete observations. As no other assumptions on space exists, the distance traveled by a robot in a cycle is unpredictable.

2.2 The Pattern Formation Problem

We study the pattern formation problem for arbitrary geometric patterns, where a pattern \( \mathcal{P} \) is a set of points \( p_0, \ldots, p_n \) (given by their Cartesian coordinates) in the plane. The pattern is known initially by all robots in the system. Initially, the robots are in arbitrary positions, with the only requirement that no two robots be in the same position, and that, of course, the number of points prescribed in the pattern and the number of robots are the same.

Let a configuration (of the robots) at time \( t \) be a set of robots’ positions at time \( t \), one position per robot, with no position occupied by more than one robot. Given a pattern \( \mathcal{P} \) and a configuration \( \mathcal{C} \), let \( P_i^t \) be the set of positions of the robots as viewed in the local coordinate system of the robot \( r_i \). The robots are said to have formed \( \mathcal{P} \), if there exists a transformation \( \mathcal{T} \), where \( \mathcal{T} \) can be translation, rotation, scaling, or flipping into mirror position, such that, \( \forall i, \mathcal{T}(P_i^t) = \mathcal{P} \). In other words, the final positions of the robots must coincide with the points of the input pattern, where the formed pattern may be translated, rotated, scaled, and flipped into its mirror position with respect to the input pattern \( \mathcal{P} \) in each local coordinate system. A final configuration for \( \mathcal{P} \) is a configuration of the robots in which the robots form the desired pattern \( \mathcal{P} \). The arbitrary pattern formation problem is the problem of devising an algorithm which will lead the robots to a final configuration in a finite number of moves for any pattern and any configuration. Consider a set of patterns \( \mathcal{X} \). Given an arbitrary initial configuration of the robots and an arbitrary pattern \( \mathcal{P} \in \mathcal{X} \), a pattern formation algorithm for \( \mathcal{P} \) is an oblivious deterministic algorithm that brings the robots in robot on any of the observed points, included the position where the observing robot is).
the system to a final configuration for \( P \) in a finite number of cycles. We say that a pattern formation algorithm is collision-free, if, at any point in time \( t \), there are no two robots that occupy the same position in the plane at \( t \).

Another problem that we will refer to in the following is the leader election problem: the robots in the system are said to elect a leader if, after a finite number of cycles, all the robots deterministically agree on (choose) the same robot \( l \), called the leader. A deterministic algorithm that lets the robots in the system elect a leader in a finite number of cycles, starting from any configuration, is called a leader election algorithm.

In the following we will denote by \( r.x \) and \( r.y \) the coordinates of robot \( r \). With calligraphic letters (e.g., \( C \)), we indicate regions of the plane, and by \( | \cdot | \) the number of robots in a given region. With capital letters (e.g., \( L \)) we indicate lines in the plane, and by \( | \cdot | \) the number of robots on a given line. Due to space restrictions, some of the proofs will be omitted.

### 2.3 Basic Properties

First notice that two robots can trivially form any pattern (the only possible pattern is a segment). Therefore, we will assume \( n > 2 \), with \( n \) the number of robots in the system.

**Theorem 1** \cite{2} The arbitrary pattern formation problem is solvable by \( n > 2 \) anonymous robots without chirality if and only if \( n \) is odd.

Interestingly, the proof of the unsolvability for \( n \) even relies only on anonymity (and lack of chirality) and not on obliviousness or asynchrony. On the other hand, the arbitrary pattern formation algorithm for \( n \) odd is both oblivious and asynchronous. To better understand the negative result for an even number of robots, we will re-prove it in terms of leader-election.

**Theorem 2** When \( n \) is even, there exists no deterministic algorithm that solves the leader election problem without chirality.

which yields Theorem 1 as a corollary.

Since an arbitrary pattern can not be formed by an even number of robots, we are now interested in determining which class of patterns, if any, can be formed in this case. Thus, from now on, we will assume that the number \( n \) of robots in the system is even.

### 3 Unformable Patterns

In this section we describe a class of pattern that cannot be formed by an even number of anonymous robots with limited agreement. A pattern \( P \) is symmetric if
Figure 1: (a) An unachievable asymmetric pattern. In this example, the sorted sequence of pairs of robots from the proof of Lemma 3 is the following: \((r_1, r_2), (r_1, \emptyset), (r_1, \emptyset), (r_3, \emptyset), (r_3, \emptyset), (r_4, \emptyset), (r_5, r_7)\). In this case \(r_0\) would be elected as the leader. (b) An achievable pattern with one empty axis. (c) An unachievable pattern. (d) An achievable pattern that has three empty axes. Note that this pattern has also axes of symmetry passing through vertices. In this case, following the procedure in Footnote 6, the routine \texttt{Choose(\mathbb{P})}\ of Algorithm 1 chooses \(S_2\).

it has at least one axis of symmetry \(S\); that is, for each \(p \in \mathbb{P}\) there exists exactly another point \(p' \in \mathbb{P}\) such that \(p\) and \(p'\) are symmetric with respect to \(S\) (see Figure 1.b, c and d).

The proof of the unsolvability result of Theorem 2 is useful to better understand which kind of patterns can not be formed, hence which kind of pattern formation algorithm can not be designed. In fact, the ability to form a particular type of patterns would imply the ability to elect a robot in the system as the leader. More formally,

**Theorem 3** There exists no pattern formation algorithm that lets the robots in the system form (a.) an asymmetric pattern, or (b.) a symmetric pattern that has all its axes of symmetry passing through a vertex.

**Proof.**

**Part a.** By contradiction, let \(\mathcal{A}\) be a pattern formation algorithm, and let \(\mathbb{P}\) be an arbitrary asymmetric pattern of \(n\) points. We now show that \(\mathcal{A}\) is a leader election algorithm. Let \(\Psi\) be the final configuration after they execute the algorithm, starting from an arbitrary initial configuration. Since the robots in the system agree on the direction and orientation of the \(Y\) axis, it is possible for them to elect a leader. In fact, let \(O_1\) and \(O_2\) be respectively the vertical axis passing through the outermost robots in \(\Psi\) (all the robots must agree on these two axis, since they agree on the orientation of \(Y\)), and let \(K\) be the vertical axis equidistant from \(O_1\) and \(O_2\) (e.g., see Figure 1.b). \(K\) splits the plane in two regions, \(S_1\) and \(S_2\). If some robots are on \(K\), the
highest on $K$ can be elected as a leader. Suppose that no robot is on $K$. We can distinguish two cases:

1. $|S_1| \neq |S_2|$. In this case, the robots can agree on the most populated region as the positive side of $X$; hence, starting from any initial configuration, it is possible to elect a leader (e.g., the topmost rightmost one).

2. $|S_1| = |S_2|$. In this case, for each robot $r_i \in S_1$, we build a pair $(r_i, x)$, $x \in S_2 \cup \emptyset$, as follows. Let $h(r)$ indicates the height of robot $r$, and $p_K(r)$ indicates the proximity of robot $r$ to $K$, that is the horizontal distance between $r$ and $K$. If there exists $r_j \in S_2$ such that $h(r_i) = h(r_j)$, and $p_K(r_i) = p_K(r_j)$, then $x = r_j$; otherwise $x = \emptyset$. Analogously, we build pairs for each $r_j \in S_2$. Given that $(r_i, r_j)$ is defined if and only if $(r_j, r_i)$ is defined, we can sort all the pairs in descending order, with respect to the height and the proximity of the robots to $K$. Namely, (e.g., see in Figure 1.b):

$$
(r_i, \emptyset) > (r_j, \emptyset) \iff h(r_i) > h(r_j) \lor (h(r_i) = h(r_j) \land p_K(r_i) < p_K(r_j))
$$

$$
(r_i, \emptyset) > (r_j, r_k) \iff h(r_i) > h(r_j) \lor (h(r_i) = h(r_j) \land p_K(r_i) < p_K(r_j))
$$

$$
(r_i, r_j) > (r_k, \emptyset) \iff h(r_i) > h(r_k) \lor (h(r_i) = h(r_k) \land p_K(r_i) < p_K(r_k))
$$

$$
(r_i, r_j) > (r_k, r_h) \iff h(r_i) > h(r_k) \lor (h(r_i) = h(r_h) \land p_K(r_i) < p_K(r_h))
$$

We observe that the set of pairs obtained is independent from the orientation of the $X$ axis; moreover, since $\Psi$ is asymmetric w.r.t $K$ by hypothesis, there must exist at least a pair with an $\emptyset$. It follows that we can elect as a leader the robot in the first pair that has $\emptyset$ as an element.

Hence, $A$ would be a leader election algorithm, contradicting Theorem 2.

**Part b.** By contradiction, suppose there exists a pattern formation algorithm $A$ that lets the robots form a symmetric pattern $\mathbb{P}$ that has all its axes of symmetry passing through some vertices in $\mathbb{P}$, starting from an arbitrary initial configuration. After the robots run $A$, they are in a final configuration $\Psi$ whose positions correspond to the vertices of $\mathbb{P}$ (up to scaling and rotation); hence, $\Psi$ must be symmetric with all its axes of symmetry passing through some vertices (robots’ positions). We distinguish two cases.

1. $\Psi$ is not symmetric w.r.t. any axis $Y'$ parallel to $Y$. In this case, the same argument of Part a. can be used to conclude that a leader can be elected.
2. \( \Psi \) is symmetric w.r.t. some \( Y' \) parallel to \( Y \). Since by hypothesis \( Y' \) must pass through a vertex, a leader can be elected (e.g., the topmost robot on \( Y' \)).

Hence, \( A \) would be a leader election algorithm, contradicting Theorem 2.

\[ \Box \]

Let us call \( \Xi \) the class containing all the arbitrary patterns, and \( \mathcal{P} \subset \Xi \) the class containing only patterns with at least one axis of symmetry not passing through any vertex (e.g., see Figures 1.b and 1.d); let us call empty such an axis. Theorem 3 states that if \( \mathcal{P} \in \Xi \setminus \mathcal{P} \), then \( \mathcal{P} \) can not be formed; hence, according to Part b. of the previous theorem, the only patterns that might be formed are symmetric ones with at least one empty axis. In the following, we prove that all these patterns can actually be formed. In particular, we present an algorithm that lets the robots form exactly these kind of patterns, if local rotation of the pattern is allowed.

## 4 Forming Formable Patterns

### 4.1 The Algorithm

In this section we present an algorithm that let the robots form symmetric patterns with at least one empty axis. The idea behind the algorithm is as follows. First, the robots compute locally an empty axis say \( S \), of the input pattern \( \mathcal{P} \), and then rotate \( \mathcal{P} \) so that \( S \) is parallel to the common understanding of the orientation of \( Y \). Second, the robots elect the two topmost outermost\(^5\) robots in the observed robots’ positions, \( \text{Outer}_1 \) and \( \text{Outer}_2 \). If this is not possible (i.e., all the robots are on the same vertical axis), the second topmost robot on this axis moves to its right, so that \( \text{Outer}_1 \) and \( \text{Outer}_2 \) can be correctly computed. Then, \( \text{Outer}_1 \) and \( \text{Outer}_2 \) move until they are at the same height: these two robots will never move again. When this happens, the vertical axis \( K \) is computed; it is the median between the vertical axis passing through the positions of \( \text{Outer}_2 \) and \( \text{Outer}_2 \); \( K \) splits the plane in two halves, called \text{Left} and \text{Right}. At this point, it is possible to compute the set of final positions of the robots: these are the positions that the robots must occupy in order to correctly solve the problem. In order to compute these positions, \( \mathcal{P} \) is scaled with respect to the distance between \( \text{Outer}_1 \) and \( \text{Outer}_2 \), and is translated with respect to the positions of these two robots. We note that, by definition of \( \mathcal{P} \), the number of final positions in \text{Left} and in \text{Right} is \( n/2 \). The robots’ positions in each half can be sorted using the ordering defined in the proof of Theorem 3. Thus, the first \( n/2 \) robot in \text{Left} are directed towards final positions in \text{Left}, and the first \( n/2 \) robots in \text{Right} towards final positions in \text{Right}. If in one half there are more robots than final destination, the \( \text{extra} \) robots are directed towards \( K \).

\(^5\)That is, among the outermost robots, the two topmost ones.
Once there are no extra robots neither in Left nor in Right, the robots on K are directed, from the topmost to the bottommost, towards the final destinations that do not have any robots on them yet.

The routine Choose(\texttt{P}) locally chooses an empty axis of symmetry in the input pattern \texttt{P}; since this is a local operation, and \texttt{P} is the same for all the robots, every robot can be made to choose the same axis of symmetry.\footnote{For instance, starting from the point \{1, 0\} on the unit circle centered in the origin of the local coordinate system, the first empty axis that is hit moving counterclockwise (according to the local orientation of the \textit{X} axis), after having translated the empty axes in such a way that they pass through the origin. In the example depicted in Figure 1.e, the axis \textit{S}_2 would be chosen.}

Rotate(\texttt{P},S) locally rotates \texttt{P} in such a way that the axis of symmetry \textit{S} chosen with Choose(\texttt{P}) is parallel to the \textit{Y} axis. The rotation is (locally) performed clockwise.

Pattern Length(\texttt{P}) returns the horizontal length of \texttt{P} according to the local unit distance, measured as the distance between the two outermost vertical axes tangent to \texttt{P}.

As already stated previously, the algorithm locates the two outermost and topmost robots, so that the input pattern can be translated and scaled with respect to them. Unfortunately, if all the robots are on the same vertical axis \textit{H}, these two robots can not be located. In Line 5, the algorithm handles this case. In particular, routine Same Vertical Axis(\textit{H}) let one of the robots on \textit{H} move to its right, so that there are exactly two outermost topmost robots. Namely, we have

\[
\text{Same Vertical Axis(\textit{H})}
\]

\[
d := \text{dist}\left(\text{top(\textit{H})}, \text{bottom(\textit{H})}\right);
\]

\[
\text{Case } [\textit{H}]
\]

\[
\cdot n
\]

\[
\text{If I Am The Second Topmost Robot On } \textit{H} \text{ Then}
\]

\[
5: \quad p := \text{Point To My Right At Horizontal Distance } d \text{ From } K;
\]

\[
\text{Move}(p).
\]

\[
\text{Else}
\]

\[
\text{do nothing()}.\nonumber
\]

\[
\cdot n - 1
\]

\[
10: \quad r := \text{Robot Not On } \textit{H} ;
\]

\[
\text{If I Am } r \text{ Then}
\]

\[
15: \quad \text{If I Am Not At Horizontal Distance } d \text{ from } \textit{H} \text{ Then}
\]

\[
p := \text{Closest Point To Me At Horizontal Distance } d \text{ From } \textit{H};
\]

\[
\text{Move}(p).
\]

\[
\text{Else}
\]

\[
\text{return}.
\]

\[
\text{Else}
\]

\[
\text{If } r \text{ Is Not At Horizontal Distance } d \text{ From } \textit{H} \text{ Then}
\]

\[
\text{do nothing}.
\]

\[
20: \quad \text{Else}
\]

\[
\text{return}.
\]
Algorithm 1 One axis direction and orientation, \( n \) even

**Input:** An arbitrary pattern \( \mathcal{P} \) described as a sequence of points \( p_1, \ldots, p_n \), given in lexicographic order. \( \mathcal{P} \) is symmetric and has at least one empty axis.

1. \( S := \text{Choose}(\mathcal{P}); \)
2. \( P := \text{Rotate}(\mathcal{P}, S); \)
3. \( P_{\text{Length}} := \text{Pattern Length}(\mathcal{P}); \)
4. \( H := \text{Leftmost Vertical Axis With More Robots On It}; \)
5. If \( (|H| = n \text{ Or } |H| = n - 1) \) Then
   - \( \text{Same Vertical Axis}(H); \)
   - \( (\text{Outer}_1, \text{Outer}_2) := \text{Outer Most}(); \)
   - If \( \text{Outer}_1 \neq \text{Outer}_2 \) Then
     - \( \text{Fix Outermosts}(\text{Outer}_1, \text{Outer}_2). \)
6. Else
   - If I Am \( \text{Outer}_1 \text{ Or } \text{Outer}_2 \) Then
     - \( \text{do nothing}(); \)
   - Else
     - \( K := \text{Median Axis}(\text{Outer}_1, \text{Outer}_2); \)
     - \( \text{Final Positions} := \text{Find Final Positions}(K, P, S, P_{\text{Length}}, \text{Outer}_1, \text{Outer}_2); \)
     - If I Am On One Of The \( \text{Final Positions} \) Then
       - \( \text{do nothing}(); \)
     - \( (\text{Left}, \text{Right}) := \text{Sides}(K); \)
8. If I Am In \( \text{Left} \text{ Or } \text{Right} \) Then
   - \( \text{MySide} := \text{My Side}(K); \)
   - \( \text{Free Points} := \{ \text{Final Positions in MySide with no robots on them} \}; \)
   - \( \text{Free Robots} := \{ \text{Robots' posit. in MySide not on Final Positions} \}; \)
   - If \( \text{Free Points} \neq 0 \) Then
     - \( \text{Go To Points}(\text{Free Robots}, \text{Free Points}); \)
     - Else
       - \( \text{Choose On}_K(\text{Free Robots}, K); * \text{I am a Free Robots but there are no Free Points in MySide} * \)
   - If I Am On \( K \) Then
     - If There Are Robots In \( (\text{Left} \cup \text{Right}) \) Not On \( \text{Final Destinations} \) Then
       - \( \text{do nothing}(); \)
     - Else
       - \( r := \text{Highest}(K); \)
       - \( p := \text{Point In Final Positions Closest To } r \text{ With No Robot On It}; \)
     - \( \text{Go To Points}([r], \{p\}). \)

where \( \text{Move}(p) \) terminates the local computation of the calling robot and moves it towards the point \( p \), using a straight movement. In other words, the distance \( d \) between the topmost (returned by \( \text{top}(H) \)) and the bottommost (returned by \( \text{bottom}(H) \)) robot on \( H \) is computed by \( \text{dist}(\text{top}(H), \text{bottom}(H)) \); if there are exactly \( n \) robots on \( H \), the second topmost robot \( r \) on \( H \) moves to its right until it is at a horizontal distance \( d \) from \( K \) (while \( r \) is moving, the case \( |H| = n - 1 \) forces all the other robots to stay still until \( r \) is at distance \( d \) from \( K \)).

\( \text{Outermost()} \) returns the current topmost outermost robots in the world. Since there is no agreement on the orientation of the \( X \) axis, it returns two robots, \( \text{Outer}_1 \) and \( \text{Outer}_2 \).

Then, the algorithm calls the routine \( \text{Fix}_\text{Outermosts}(\text{Outer}_1, \text{Outer}_2) \), that moves \( \text{Outer}_1 \) and \( \text{Outer}_2 \) until they reach the same height. Until this happens, all the other robots are not allowed to move.

The function \( \text{Median}_\text{Axis}(\text{Outer}_1, \text{Outer}_2) \) returns the vertical axis \( K \), which is the median axis between the vertical axes passing through \( \text{Outer}_1 \) and \( \text{Outer}_2 \).

\( \text{Find}_\text{Final}_\text{Positions}(K, P, S, \text{Pattern}_\text{Length}, \text{Outer}_1, \text{Outer}_2) \) executes the scaling and the translation of \( \mathbb{P} \) with respect to the positions of \( \text{Outer}_1 \) and \( \text{Outer}_2 \). These positions are computed in the following way: \( K \) is viewed as the empty axis \( S \) computed in Line 1; furthermore, by definition of \( \mathbb{P} \) and because of the way it has been rotated in Line 2, there are exactly two outermost topmost points in \( \mathbb{P} \) that are symmetric with respect to \( S \): these two points are viewed as \( \text{Outer}_1 \) and \( \text{Outer}_2 \) (which are symmetric with respect to \( K \)). The common scaling of the input pattern is defined by identifying \( \text{Pattern}_\text{Length} \) with the horizontal distance between \( \text{Outer}_1 \) and \( \text{Outer}_2 \) (that are at the same height and already in their final positions).

The routine \( \text{Sides}(K) \) returns two sets, each containing the positions of robots currently lying in the two halves in which the plane is split by \( K \); in particular, it returns respectively the robots’ positions on the \textit{Left} and on the \textit{Right} of \( K \), according to the local orientation of the calling robot’s \( X \) axis. \( \text{My}_\text{Side}(K) \) returns the half of the plane where the calling robot currently lies.

\( \text{Go}_\text{To}_\text{Points}(\text{Free}_\text{Robots}, \text{Free}_\text{Points}) \) chooses the robot \( r \) in \( \text{Free}_\text{Robots} \) that is closest to a point in \( \text{Free}_\text{Points} \), say \( p \), and moves \( r \) in such a way to avoid collisions. In fact, if \( r \) were to move following a straight line towards its destination \( p \), it might collide with a robot on this path. Should this be the case, the routine appropriately chooses an intermediate destination point maintaining the invariant that \( r \) is the robot in \( \text{Free}_\text{Robots} \) closest to a point in \( \text{Free}_\text{Points} \). Namely (refer to Figure 2.a).

\( \text{Go}_\text{To}_\text{Points}(\text{Free}_\text{Robots}, \text{Free}_\text{Points}) \)

\( (r, p) := \text{Minimum}(\text{Free}_\text{Robots}, \text{Free}_\text{Points}); \)

\( \text{If } r \text{ Am } r \text{ Then} \)

\( \text{If No Robots Is On The Line Passing Through } r \text{ And } p \text{ Then} \)

\( \text{Move}(p). \)
Figure 2: (a) Routine \texttt{GoToPoints()} determines the destination point for \( r \). The empty circles represent the points in \texttt{Avoid}, and the grey one represents the robot that does not allow \( r \) to perform a straight movement towards \( p \); hence routine \texttt{ClosestIntersection()} is called in this case. The thick line is the path followed by \( r \) (in two cycles) to reach \( p \). (b) Routine \texttt{ChooseOnK()} determines the destination point for \( r \) on \( K \). The thick line is the path followed by \( r \) to reach \( K \).

5: \textbf{Else}  
   \[ L := \text{Line Passing Through } r \text{ and } p; \]  
   \[ L' := \text{Line Orthogonal To } L; \]  
   \[ C_p := \text{Circle Centered In } p \text{ Having Radius } r_p; \]  
   \[ C'_p := \text{Half Of } C_p \text{ Delimited By } L' \text{ That Contains } r; \]  
10: \textbf{Avoid := Robots’ Positions Inside } \( C'_p; \)  
   \[ p'' := \text{ClosestIntersection}(p,L'',\text{Avoid,FreePoints}); \]  
   \textbf{If } \( p'' \) Is Not Inside \( C_p \text{ Then} \)  
   \[ p'' := \text{Point On } L' \text{ At Distance } r_p/2 \text{ From } p; \]  
   \{ \text{In this way, } p'' \text{ is always chosen on } L' \text{ and inside } C_p \}  
15: \textbf{Move}(p'). \]  
\textbf{Else}  
   \textbf{do nothing().}  

where \texttt{Minimum(FreeRobots,FreePoints)} returns the pair \((r,p)\) such that

\[
    r_p = \min_{f_j \in \text{FreeRobots}} \left\{ \min_{f_j \in \text{FreePoints}} f_j f_p \right\}
\]

If more than one robot has minimum distance, the topmost robot that is closest to \( K \) is chosen. The core of this routine is in \texttt{ClosestIntersection()}, that looks for a point \( p'' \) on \( L' \) such that inside the triangle \( \Delta \) passing through \( r, p, \) and \( p'' \) there are no robots; hence, \( r \) can move inside \( \Delta \) avoiding collisions. Furthermore, \texttt{ClosestIntersection()} checks also that \( p'' \) is such that, while \( r \) moves towards it, \( p \) remains the point in \texttt{FreePoints} closest to \( r \).
If in MySide there are more robots than final positions, the extra robots are directed towards $K$ by invoking routine ChooseOnK() in Line 27. This routine ensures that the movements towards $K$ are done without collisions (refer to Figure 2.b). In particular, ChooseOnK(FreeRobots, K)

If I Am The Topmost And Closest To $K$ in FreeRobots Then
$p :=$ Intersection Between $K$ And Horizontal Line Passing Through My Position;
If No Robot Is On The Line Passing Through My Position And $p$ Then
Move ($p$).
Else
$L :=$ Vertical Line Passing Through My Position;
$R :=$ Portion Of The Plane Above My Position And Delimited By $K$ And $L$;
Avoid := \{Robots’ Positions Inside $R$\} \cup \{Robots’ Positions On $K$ Above $p$\};
Intersections := $\emptyset$
For All $p' \in$ Avoid Do
$L' :=$ Line Passing Through My Position And $p'$;
Intersections := Intersections\{Intersection Between $K$ And $L'$\};
End For
$p' :=$ Topmost Point In Intersections;
$p'' :=$ Point On $K$ Above $p'$ At Distance $\delta$ From $p'$.
Move ($p''$).
Else
do nothing().

In other words, the robot $r$ chooses a path the goes above all the robots that are in the region $R$, delimited by $L$ (the vertical line passing through the position of the calling robot) and $K$, maintaining the invariant to remain the (closest to $K$) topmost robot in FreeRobots.

When all the robots in Left and Right are on Final Positions, all the robots on $K$, if any, are directed sequentially, from the topmost to the bottommost, towards the available final positions. In particular, the topmost robot on $K$ chooses the closest point in FinalPositions.

4.2 Correctness

We shall call an even agreement configuration one in which (i) all robots agree on the position of a vertical axis $K$ in the plane, (ii) each robot is either on one of the Final Positions or on $K$, and (iii) there is at most one robot on each of the Final Positions. Moreover, we define the cardinality of an even agreement configuration as the number of robots on $K$. In the following we will denote by sides the two halves in which the plane is divided by $K$. We first show that Algorithm 1 lets the robots reach an even agreement configuration in a finite number of cycles,
while avoiding collisions between robots.

**Lemma 1** If the robots are not in a final configuration, in a finite number of cycles the robots agree on two topmost outermost robots, Outer₁ and Outer₂. Let O₁ and O₂ be the vertical axes passing through the positions of Outer₁ and Outer₂, respectively. In a finite number of cycles Outer₁ and Outer₂ will be at the same height, and all the robots will agree on the position of the vertical axis K that is median between O₁ and O₂. Until this happens, any collision is avoided.

**Proof.** After having chosen an axis of symmetry S on Line 1 of the algorithm⁷, the robot locally rotates P in such a way that S becomes parallel to Y. We note that, since P is symmetric with respect to S, and S is an empty axis, there are exactly two topmost outermost points in P. If the robots are all on the same vertical axis H, routine Same_Vertical_Axis (H) is called. In particular, let d be the distance between the topmost and the bottommost robot on H. The second topmost robot r on H moves at a distance d to its right, while all the other robots do not move. While r is moving, |H| = n - 1 and, according to Line 19 of the routine, all the robots do not move until r, in a finite number of movements, is at distance d from K. After this, the two outermost and highest robots in the world are localized (Outer₁ and Outer₂ at Line 7), and until they are at the same height⁸ all the other robots do not move (Lines 3 and 9 in routine Fix_Outermosts()). Once they reach the same height (Line 10), Outer₁ and Outer₂ will never move again (Line 12). By construction, O₁ ≠ O₂, and there are no robots on O₁ and O₂ that are above Outer₁ or Outer₂. Therefore, since according to routine Fix_Outermosts() Outer₁ and Outer₂, if they move, always move up and on O₁ and O₂, their movements can not produce any collision. After Outer₁ and Outer₂ reach the same height, all the robots can agree on the axis K that is in the middle between the vertical axis passing through Outer₁ and the vertical axis passing through Outer₂ (Line 14). Since Outer₁ and Outer₂ executed a finite number of cycles to reach their final positions, by Assumptions A1 and A2, the agreement on K is reached in a finite number of cycles. □

**Lemma 2** If the robots are not in a final configuration nor in an even agreement configuration, they will reach an even agreement configuration in a finite number of cycles, and without collisions.

Moreover, we can prove the following

**Lemma 3** Given an even agreement configuration of cardinality greater or equal to one, within finite time another even agreement configuration of smaller cardinality will be reached, avoiding any collisions.

---

⁷ All the Lines refer to Algorithm 1, if not otherwise stated.

⁸ We recall that we can talk about same heights because all the robots agree on the direction of Y, hence they can commonly agree on this.
Therefore, by Lemmas 2 and 3, we can state the following

**Theorem 4** Algorithm 1 is a collision-free pattern formation algorithm for $\mathcal{P} \in \mathcal{P}$. 

**Corollary 1** An even number of autonomous, anonymous, oblivious, mobile robots that agree on the direction and orientation of $Y$ axis, can form a pattern $\mathcal{P}$ if and only if $\mathcal{P} \in \mathcal{P}$. 

5 Remarks on Rotation

In Sections 3 and 4.1, we have provided a characterization of the class of patterns which can be formed by an even number of anonymous robots, provided they have agreement on the direction and orientation of the $Y$ axis. This characterization assumes that the robots can locally rotate the input pattern. Should the robots be incapable to perform this operation, the characterization is different; not surprisingly, the class of achievable patterns is smaller. Let $\mathcal{P}' \subset \mathcal{P}$ be the class of symmetric pattern with at least one empty axis, and with no empty axis parallel to $Y$.

**Theorem 5** There exists no pattern formation algorithm that does not allow local rotation of the input pattern, and that lets the robots form a symmetric pattern $\mathcal{P} \in \mathcal{P}'$. 

**Proof.** By contradiction, let $\mathcal{A}$ be an algorithm that, starting from an arbitrary initial configuration, let the robots form a pattern $\mathcal{P} \in \mathcal{P}'$. Let $\Psi$ be the final configuration of the robots for $\mathcal{P}$ after they execute $\mathcal{A}$. Since no local rotation is allowed, also $\Psi$ is symmetric with no empty axis parallel to $Y$. Let $O_1$ and $O_2$ be the vertical axes passing through the two outermost robots, and $K$ the vertical axis median between $O_1$ and $O_2$. If $K = O_1 = O_2$, then all the robot are on $K$, hence a leader can be elected (e.g., the topmost robot on $K$), thus contradicting Theorem 2. Otherwise, if $\Psi$ is symmetric with respect to $K$, then there must be at least one robot on $K$ (by hypothesis, $\Psi$ has no empty axis parallel to $Y$); hence, the topmost of these robots can be elected as leader, contradicting Theorem 2. Therefore, $\Psi$ is not symmetric with respect to $K$: also in this case a leader can be elected (e.g., following an approach similar to the one used in the proof of Theorem 3.a), thus contradicting again Theorem 2. □ 

As a concluding remark, we note that skipping the $\text{Rotate} (\mathcal{P})$ at Line 2 in Algorithm 1, we have a pattern formation algorithm that does not make use of local rotation and allows the formation of a symmetric pattern that has at least one empty axis that is parallel to $Y$. Hence, we can state the following
Corollary 2  An even number of autonomous, anonymous, oblivious, mobile robots that agree on the direction and orientation of Y axis, can form a pattern $\mathbb{P}$ if and only if $\mathbb{P} \in \mathbb{P} \setminus \Psi'$, when no local rotation of $\mathbb{P}$ is allowed.

Acknowledgements

This work has been partially supported by the Natural Science and Engineering Research Council of Canada, by the Swiss National Science Foundation, and by Nortel Networks.

References


Paola Flocchini  is an Associate Professor at the School of Information, Technology and Engineering of the University of Ottawa. E-mail: flocchin@site.uottawa.ca

Giuseppe Prencipe is a Ph.D. student at the Computer Science Department of the University of Pisa. E-mail: prencipe@di.unipi.it

Nicola Santoro is a Full Professor at the School of Computer Science of the Carleton University. E-mail: santoro@scs.carleton.ca

Peter Widmayer is a Full Professor at the Theoretische Informatik of the ETH Zürich. E-mail: pw@inf.ethz.ch
Characterization of Networks Supporting Multi-dimensional Linear Interval Routing Scheme

YASHAR GANJALI
University of Waterloo, Canada

Abstract
An Interval routing scheme (IRS) is a well-known, space efficient routing strategy for routing messages in a distributed network. In this scheme, each node of the network is assigned an integer label and each link at each node is labeled with an interval. The interval assigned to a link $e$ at a node $v$ indicates the set of destination addresses of the messages which should be forwarded through $e$ at $v$. A Multi-dimensional Interval Routing Scheme (MIRS) is a generalization of IRS in which each node is assigned a multi-dimensional label (which is a list of $d$ integers for the $d$-dimensional case). The labels assigned to the links of the network are also multi-dimensional (a list of $d$ 1-dimensional intervals). The class of networks supporting linear IRS (in which the intervals are not cyclic) is already known for the 1-dimensional case [5]. In this paper, we generalize this result and completely characterize the class of networks supporting linear MIRS (or MLIRS) for a given number of dimensions $d$. We show that by increasing $d$, the class of networks supporting MLIRS is strictly expanded. We also give a characterization of the class of networks supporting strict MLIRS (which is an MLIRS in which the intervals assigned to the links incident to a node $v$, does not contain the label of $v$).

Keywords
Computer networks, interval routing schemes, graph theory, multi-dimensional, characterization.

1 Introduction
One of the most fundamental tasks in any network of computers is routing messages between pairs of nodes. The classical method used for routing messages in a network is to store a routing table at each node of the network. A routing table

*This work has been supported in part by NSERC.
has one entry for each destination address which indicates which of the adjacent links should be used to forward the message towards that destination.

Each routing table requires $O(n)$ space in an $n$-node network, which is not efficient (and even feasible) for large networks of computers. The methods to reduce the amount of space needed at each node have been intensively studied and there are many techniques to compress the size of routing tables [6, 7, 1, 12]. The general idea is to group the destination addresses that correspond to the same outgoing link (at a node), and to encode the group so that it is easy to verify if a given destination address is in the group or not. A well-known solution is to use intervals as groups of destination addresses.

In an Interval Routing Scheme (IRS), which was originally introduced by Santoro and Khatib [11], each node of the network is assigned an integer label taken from $\{1, 2, \ldots, n\}$ and each link of the network at each node is assigned an interval. Routing messages is completed in a distributed way. At each intermediate node $v$, if the label of the node equals the destination address, dest, the routing process ends. Otherwise, the message is forwarded through a link labeled by an interval $I$, such that $dest \in I$. Clearly, this method requires $O(l)$ space at each node ($l$ is the number of links at the node), which is an efficient memory allocation.

A Linear Interval Routing Scheme (LIRS) is an IRS in which the intervals are not cyclic. Also, a Strict Interval Routing Scheme (SIRS) is an IRS in which the interval assigned to a link $e$ at a node $v$ does not contain the label of $v$. A Strict and Linear Interval Routing Scheme (SLIRS) is an IRS which is both linear and strict. If we assign $k$ intervals to each link of the network we will have a $k$-IRS (respectively, $k$-LIRS, $k$-SIRS, and $k$-SLIRS). Gavoille has done a survey of results concerning this method [9].

It has been proved that any network supports an SIRS and therefore an IRS [11, 13]. The class of networks which support LIRS and SLIRS have also been characterized by Fraigniaud and Gavoille which excludes a large class of networks [5]. They define a class of graphs called lithium graphs and show that a network supports an LIRS if and only if its underlying graph is not a lithium graph. They also show that a network supports an SLIRS if and only if its underlying graph is not a weak lithium graph.

A very interesting extension of IRS is a Multi-dimensional Interval Routing Scheme (MIRS) in which the labels assigned to the nodes are elements from $\mathbb{N}^d$ (in the $d$-dimensional case) and each link is labeled with a $d$-tuple $([a_1, b_1], [a_2, b_2], \ldots, [a_d, b_d])$ of intervals, $a_i, b_i \in \mathbb{N}$, for $1 \leq i \leq d$ [4]. The routing process in an MIRS is quite similar to the routing process in 1-dimensional IRS.

A network is said to be in $\langle k, d \rangle$-MIRS or support $\langle k, d \rangle$-MIRS if there is a $d$-dimensional MIRS with $k$ intervals in each link such that for any pair of nodes $s$ and $t$, the message originating from $s$ eventually reaches $t$. The classes $\langle k, d \rangle$-MLIRS and $\langle k, d \rangle$-MSLIRS are defined similarly. The only known classes of networks which support different variations of MIRS are specific interconnection networks such as rings, grids, tori, hypercubes and chordal rings. In this paper,
we will investigate the problem of characterizing classes of networks supporting MIRS. We give a complete characterization of the class of networks supporting $\langle 1, d \rangle$-MLIRS and $\langle 1, d \rangle$-MSLIRS. We show that the class of networks supporting $\langle 1, d \rangle$-MLIRS ($\langle 1, d \rangle$-MSLIRS) is a strict subset of the class of networks supporting $\langle 1, d + 1 \rangle$-MLIRS ($\langle 1, d + 1 \rangle$-MSLIRS) and therefore, increasing the number of dimensions in an MLIRS (MSLIRS) increases the power of the routing scheme.

The rest of this paper is organized as follows: first, we will introduce some definitions and preliminaries in Section 2. In Section 3 we will characterize the class of graphs supporting $\langle 1, d \rangle$-MLIRS. Then, in Section 4, based on the arguments of the previous section, we will give a characterization for graphs supporting $\langle 1, d \rangle$-MSLIRS. Finally, in Section 5 we will conclude and give a list of open problems.

2 Preliminaries

Throughout this paper, a network is modeled by a graph $G = (V, E)$. The set $V$ of vertices of the graph represents nodes in the network and the set $E$ of edges represents the links between the nodes in the network. We assume that the graph is simple and does not have any self-loops. For any edge $(u, v) \in E$ we will use both $(u, v)$ and $(v, u)$ in order to assign two unidirectional labels to the edge, but the edge is assumed to be undirected. We refer the reader to standard texts for basic graph theoretic definitions [3, 14].

A graph $G$ is said to be connected if for any pair of vertices, $s$ and $t$, there is a path connecting $s$ and $t$. In this paper, we always assume that the network is connected. If removing an edge $e$ disconnects a graph $G$, $e$ is called a bridge. If a graph does not have a bridge, it is said to be edge-biconnected. Edge-biconnected components of a graph $G$ are maximal subgraphs of $G$ which are edge-biconnected.

**Observation 1** If $G_1$ and $G_2$ are two edge-biconnected components in a graph $G$, then any path $P$ connecting $G_1$ and $G_2$ goes through a unique bridge connected to $G_1$.

In the following section, we will give a characterization for the class of networks supporting a $\langle 1, d \rangle$-MLIRS.

3 Characterization of networks supporting $\langle 1, d \rangle$-MLIRS

In this section we first give some examples of graphs which do not support $\langle 1, d \rangle$-MLIRS. Using the idea behind these examples, we introduce a class of graphs which do not support $\langle 1, d \rangle$-MLIRS. Finally, we show that for any graph that is not in this class, one can always construct a $\langle 1, d \rangle$-MLIRS.
Bakker et al. [2] have proved that the graph shown in Figure 1 (a) (known as the Y graph) does not have an LIRS (which is a \( \{1,1\}\)-MLIRS). Here, we prove a similar result in the \( d \)-dimensional case. First, let us start by generalizing the definition of a Y graph.

**Figure 1:** (a) The Y graph (b) The \( Y_5 \) graph (c) A 5-windmill graph.

**Definition 1** The \( Y_k \) graph is a graph having \( 2k+1 \) vertices \( u_1, u_2, \ldots, u_k, v_1, v_2, \ldots, v_k \) and \( z \). There is an edge connecting \( u_i \) to \( v_i \), for every \( i, 1 \leq i \leq k \), and another edge connecting each \( v_i \) to \( z \), \( 1 \leq i \leq k \) (Figure 1 (b)). We call the subgraph consisting of \( u_i \) and \( v_i \) the \( i \)th wing of the graph.

The Y graph of Figure 1 (a) is a \( Y_3 \) graph by our new definition. To prove that the \( Y_3 \) graph does not have an LIRS let us assume it has an LIRS and the vertices of the graph are assigned integer labels taken from \( \{1,2,\ldots,7\} \). Since we have three wings, there is a wing, say the \( i \)th wing, which does not contain 1 or 7 (the minimum or the maximum label). Now, the interval assigned to the edge \( (v_i,z) \) at \( v_i \) must contain both 1 and 7. Therefore, this interval contains the label of \( u_i \) which is not possible.

We can prove a similar result for \( d \)-dimensional LIRS and for the \( Y_{2d+1} \) graph. In fact, we can immediately observe that if each wing of the \( Y_{2d+1} \) graph had more than just two vertices, as long as those vertices are not directly connected to the vertex \( z \) or to the vertices in other wings, the graph cannot support a \( d \)-dimensional MLIRS. In order to prove this more general statement, we define a \( k \)-windmill graph as follows.

**Definition 2** A \( k \)-windmill graph is a connected graph with \( k+1 \) connected components (not necessarily maximal) \( A_1, A_2, \ldots, A_k \) (arms of the \( k \)-windmill graph) and \( R \) (center of the \( k \)-windmill graph) such that:

(i) each component \( A_i, 1 \leq i \leq k \), has at least two vertices;
(ii) there is no edge connecting $A_i$ to $A_j$ for $1 \leq i, j \leq k$ and $i \neq j$; and

(iii) each component $A_i$, $1 \leq i \leq k$, is connected with $R$ by exactly one bridge.

Figure 1 (c) illustrates a 5-windmill graph. Obviously, by this definition, a $Y_k$ graph is also a $k$-windmill graph. Also, as Figure 1 (c) indicates, a $k$-windmill graph is a $i$-windmill graph for any $i$, $1 \leq i \leq k − 1$. This can easily be shown by expanding $R$ to include $A_{i+1}, \ldots, A_k$.

**Lemma 1** Any $(2d + 1)$-windmill graph $\notin \langle 1, d \rangle$-MLIRS.

Before proving this lemma, let us give a new definition, which will be used in the proof. We consider a set of points $P$ in $d$-dimensional space. If for any dimension $i$, $1 \leq i \leq d$, the $i$th coordinate of a point $b$ in $P$ is less than or equal to the $i$th coordinate of every other point in $P$, $b$ is called a *minimum point for the $i$th dimension*. A maximum point is defined similarly. A *boundary set* $B$ of $P$ is a minimal set of points in $P$ containing a minimum and a maximum point for each dimension $i$, $1 \leq i \leq d$, where one point can be both the minimum and the maximum point for the same or different dimensions.

![Figure 2: An example of a boundary set in 2-dimensional space.](image)

Figure 2 illustrates an example of a boundary set in 2-dimensional space. Here, $P = \{1, \ldots, 7\}$ and $\{1, 5, 7\}$ is a boundary set of $P$. The set $\{2, 5, 7\}$ is also a boundary set of $P$. We note that point 7 is the maximum point for one dimension and the minimum point for another dimension.

For any set of points in $d$-dimensional space, the number of points in any boundary set is at most $2d$. It is easy to show that if an interval contains the points in the boundary set $B$ of a set of points $P$, it contains all points in $P$. Now we can easily prove Lemma 1. In this proof, we consider the $d$-dimensional labels of vertices as points in $d$-dimensional space.

**Proof.** (Lemma 1) Let us assume, by way of contradiction, that there is a $\langle 1, d \rangle$-MLIRS for a given $(2d + 1)$-windmill graph ($d \geq 1$) and consider the boundary set $B$ of the vertices of the graph. We have at most $2d$ vertices in the boundary set $B$. Since a $(2d + 1)$-windmill graph has $2d + 1$ arms, there is an arm, say the $j$th arm,
that does not contain any vertex in the boundary set $B$. Every $d$-dimensional interval containing all of the vertices in $B$ contains all vertices of $(2d + 1)$-windmill graph as well. Thus, the interval assigned to the bridge connecting the $j$th arm to the center of the $(2d + 1)$-windmill graph, say $(u, v)$ (where $u$ is in the $j$th arm and $v$ is a vertex in the center of the graph) contains all vertices in the $(2d + 1)$-windmill graph. The $j$th wing has at least another vertex other than $u$, say $u'$. Hence, the interval assigned to the edge $(u, v)$ includes $u'$. Obviously, there is no path going through $(u, v)$ to reach $u'$, which is a contradiction.

Lemma 1 introduces a class of graphs which do not support $(1, d)$-MLIRS. In other words, it states a necessary condition for a graph to support a $(1, d)$-MLIRS. In the rest of this section we will show that this is also a sufficient condition.

Fraigniaud and Gavoille have proved that a graph supports LIRS if and only if it is not a lithium graph [5] (which is exactly the 3-windmill graph). We will use this result as the basis for an inductive construction of a $(1, d)$-MLIRS for a given graph $G$. We start with some new definitions.

**Definition 3** In a graph $G$, a chain of edge-biconnected components, or a chain for short, is a set of edge-biconnected components of $G$ with a special ordering of these edge-biconnected components, say $G_1, G_2, \ldots, G_k$, such that:

(i) for each $i, 1 \leq i \leq k - 1$, there is a bridge connecting $G_i$ to $G_{i+1}$;

(ii) $G_1$ is connected to exactly one bridge in $G$;

(iii) each edge-biconnected component $G_i$, $2 \leq i \leq k - 1$ is connected to exactly two bridges in $G$; and

(iv) The edge-biconnected component $G_k$ is connected to either one or two bridges.

![Figure 3: The dashed curves indicate edge-biconnected components in this figure. The edge-biconnected components $G_1$ and $G_2$ form a chain. The edge-biconnected components $G_1, G_2$ and $G_3$ (and not $G_4$) form a perfect chain.](image)
We call $G_1$ the head and $G_k$ the tail of the chain. Trivially if $k = 1$ then $G_1$ is both the head and the tail of the chain. A chain is said to be perfect if the tail of the chain is connected to an edge-biconnected component which is connected to more than two bridges.

### 3.1 Properties of chains and $k$-windmill graphs

In this section we review some of the properties of chains and $k$-windmill graphs. The first observation follows directly from the definition of a chain.

**Observation 2** A perfect chain in a graph $G$ is a proper induced subgraph of $G$, and the tail of a perfect chain (which is an edge-biconnected component) is connected to the rest of the graph by a bridge.

The edge-biconnected components $G_1, G_2$ and $G_3$ in the graph depicted in Figure 3 and the bridges connecting them form a chain. $G_1$ and $G_3$ are the head and the tail of this chain, respectively. This is also a perfect chain since $G_3$ (tail) is connected to an edge-biconnected component ($G_4$) which is connected to more than two bridges. As mentioned in Observation 2, $G_3$ (which is the tail of the perfect chain) is connected to the rest of the graph by a bridge. Since, $G_3$ is connected to exactly two bridges, the edge-biconnected components $G_1$ and $G_2$ does not form a perfect chain.

![Figure 4: Edge-biconnected components in a 3-windmill graph.](image)

**Lemma 2** If a graph $G$ is a $k$-windmill graph for $k \geq 3$ then it is not a chain.

**Proof.** We consider each edge-biconnected component of $G$ as a super-node. Clearly, the resulting graph is a tree (otherwise, we have a cycle which contains some bridges, a contradiction). Since, $G$ is a $k$-windmill graph ($k \geq 3$), there is
a node v in this tree such that the degree of v is at least 3 (the super-node \( G_r \) in Figure 4). In any chain, each edge-biconnected component is connected to at most 2 other edge-biconnected components. Therefore, G is not a chain. \( \square \)

**Lemma 3** Any non-trivial (having at least one vertex) graph G which is not a chain contains a perfect chain as a proper induced subgraph.

**Proof.** Similar to the proof of Lemma 2, if we consider each edge-biconnected component of G as a super-node we will have a tree. Any tree has at least one leaf. The chain starting with this leaf and going to the nearest super-node with degree at least three is a perfect chain (since G is not a chain such a super-node always exists). \( \square \)

For example, in the graph depicted in Figure 3 if we consider the chain starting from the super-node \( G_1 \) and going to \( G_3 \) (which is connected to \( G_4 \) which is of degree four) we have a perfect chain.

In constructing a \( (1,d) \)-MLIRS, we will use this lemma in the induction step to reduce the size of the graph. This reduction has a very nice property that is the heart of the main proof, which is stated in the following lemma.

**Lemma 4** If a graph G is not a chain and is not a \( k \)-windmill graph \( (k > 3) \), we can remove any perfect chain from G and the resulting graph is not a \( (k - 1) \)-windmill graph.

**Proof.** Since G is not a chain, by Lemma 3, there is a perfect chain C which is a proper induced subgraph of G. We let \( G' \) denote the graph \( G - C \). We assume, to the contrary, that \( G' \) is a \( (k - 1) \)-windmill graph. By the definition of a \( (k - 1) \)-windmill graph, \( G' \) has \( k \) disjoint sets of vertices \( A_1, A_2, ..., A_{k-1} \) and R. Since C is

![Figure 5: C and D will become arms in the k-windmill graph.](image-url)
a perfect chain, by Observation 2 its tail is connected to $G$ by a bridge. $C$ cannot be connected to $R$, otherwise $G$ must be a $k$-windmill graph. Let us assume that $C$ is connected to an edge-biconnected component, $B$, which is in the arm $A_i$ for some $i$, $1 \leq i \leq k - 1$ (Figure 5).

By the definition of a perfect chain, the edge-biconnected component $B$ is connected to at least three bridges, one connecting $B$ to $C$ and at least two other bridges connecting $B$ to some other edge-biconnected components in $G$. By Observation 1 all the paths connecting $B$ and $R$ go through one of the bridges connected to $B$, say $e$. We let $D$ be the edge-biconnected component which is connected to $B$ and is not connected to $e$.

Now, we expand $R$ to contain $B$ and all the edge-biconnected components in the arm $A_i$ except $D$. Since $G$ is a $k$-windmill graph it has $k - 2$ arms other than $A_i$. We can also consider $C$ and $D$ as two new arms. Hence, $G$ has $k$ arms and is a $k$-windmill graph, a contradiction.

3.2 Characterization

In this section we will prove the main result of this paper. First, we need to show how to convert a $d$-dimensional IRS into a $(d + 1)$-dimensional IRS.

If a graph $G$ supports a $(1, d)$-MLIRS ($\langle 1, d \rangle$-MSLIRS), we can convert the $d$-dimensional to a $(d + 1)$-dimensional one, by adding a new coordinate to the labels of vertices. The label of this coordinate is set to zero for all vertices. We also set the newly added coordinate of each interval to be $[0, 0]$. It is a trivial task to verify that this IRS routes the messages exactly like the $d$-dimensional IRS. In other words, we can expand a $d$-dimensional IRS to a $(d + 1)$-dimensional IRS.

Lemma 5 If a graph $G$ supports a $(1, d)$-MLIRS ($\langle 1, d \rangle$-MSLIRS) it also supports a $(1, d + 1)$-MLIRS ($\langle 1, d + 1 \rangle$-MSLIRS).

Now, we have all the tools we need to prove the main theorem of this section.

Theorem 1 A graph $G$ has a $(1, d)$-MLIRS if and only if it is not a $(2d + 1)$-windmill graph.

Proof. First, we show that if a graph is not in the class of $(2d + 1)$-windmill graphs, then it has a $(1, d)$-MLIRS. We use induction on $d$, the number of dimensions. Fraigniaud and Gavoille [5] have proved that if a graph $G$ is not a lithium graph, which is exactly a 3-windmill graph, then there is a 1-IRS for $G$ (a $(1, 1)$-MLIRS). This is the basis of the induction.

Let us suppose that for any $i \leq d - 1$, if a graph is not a $(2i + 1)$-windmill graph, it has a $(1, i)$-MLIRS. Now, we want to show that if a graph $G$ is not a $(2d + 1)$-windmill graph, $d > 1$, then it has a $(1, d)$-MLIRS. We first show how to label the vertices of $G$. Then, we describe how we can update intervals in each
step of the induction. Finally, we prove the correctness of such vertex and link labeling.

**Labeling vertices:**
Although $G$ is not a $(2d + 1)$-windmill graph it can be a $(2d - 1)$-windmill graph. If $G$ is not a $(2d - 1)$-windmill graph, by the induction hypothesis it has a $\langle 1, d - 1 \rangle$-MLIRS and by Lemma 5, $G$ also has a $\langle 1, d \rangle$-MLIRS, completing the proof. Hence, we can assume that $G$ is a $(2d - 1)$-windmill graph and by recalling Lemma 2, we can assume that $G$ is not a chain. Therefore, by Lemma 3, $G$ has a perfect chain, say $C_1$, as a proper induced subgraph. Since $G$ is not a $(2d + 1)$-windmill graph and $d > 1$, by applying Lemma 4 we can remove $C_1$ and the resulting graph will not be a $2d$-windmill graph. Since $2d > 3$, we can repeat these steps and remove another perfect chain, $C_2$, so that the resulting graph, $G'$, is not a $(2d - 1)$-windmill graph.

By the induction hypothesis, $G'$ has a $\langle 1, d - 1 \rangle$-MLIRS. We just need to expand this labeling to a $\langle 1, d \rangle$-MLIRS for $G$.

![Figure 6: Expanding the labels of vertices in $G'$ to labels for vertices in $G$.](image)

$C_1$ and $C_2$ are chains and therefore, by Lemma 2, they are not 3-windmill graphs. Therefore, by the induction hypothesis, there is a $\langle 1, 1 \rangle$-MLIRS for each of them. In fact, in [5], it has been proved that if a given graph is not a 3-windmill (lithium) graph, we can specify a vertex and find a labeling for the vertices such that the label of the specified vertex is 1. We find such a $\langle 1, 1 \rangle$-MLIRS for $C_1$ ($C_2$) such that the label for the vertex in $C_1$ ($C_2$) joining $C_1$ ($C_2$) to the rest of the graph $G$, say $u_1$ ($u_2$), is 1 (Figure 6).

To construct the new labeling for $G$, each vertex in $G'$ is assigned a $d$-dimensional label in which the first $d - 1$ coordinates are the same as the labels in the linear $\langle 1, d - 1 \rangle$-MIRS corresponding to $G'$ and the $d$th coordinate is 0. Figure 6 illustrates an example in which $d = 3$. The third coordinates of the labels assigned to the vertices of $G'$ are all 0, so $G'$ lies in the plane passing through
the first and the second axes. For now, we assume that the labels assigned to the vertices can have any integer value (including 0 and negative integers) as their \( d \)th coordinates. We can shift all the labels such that the \( d \)th coordinates of all labels becomes positive later.

Let \( v_1 \) and \( v_2 \) respectively denote the bridges connecting \( G' \) to \( C_1 \) and \( C_2 \) and let \( v_1 \) and \( v_2 \) be vertices of \( G' \). We will set the first \( d-1 \) coordinates of each vertex in \( C_1 \) to be equal to the first \( d-1 \) coordinates of \( v_1 \). The \( d \)th coordinates of vertex labels in \( C_1 \) are the labels assigned to vertices in the previously mentioned \( \{1,1\} \)-MLIRS. In Figure 6 the vertices in \( C_1 \) all lie on the line passing through \( v_1 \) and parallel to the \( d \)th axis.

For the vertices in \( C_2 \), we will similarly set the first \( d-1 \) coordinates of each vertex equal to the first \( d-1 \) coordinates of \( v \). If the label of a vertex \( v \) in the previously mentioned \( \{1,1\} \)-MLIRS is \( l(v) \), we assign \(-l(v)\) as the \( d \)th coordinate of the new labeling (Figure 6). Now as mentioned before, we can shift the \( d \)th coordinate of all the labels such that the \( d \)th coordinate of the vertex with minimum value becomes 1. We let \( s \) denote the amount of this shifting and \( M \) denote the maximum value in the \( d \)th coordinate of all new labels.

![Figure 7](image_url)

Figure 7: (a) Updating an interval in \( G' \) (b) Updating an interval, which includes \( u_1 \), in \( C_1 \) (\( I \) is the old interval, \( I' \) is the new one in both (a) and (b))

**Updating Intervals:**
We update intervals as follows: the first \( d-1 \) coordinates of each interval assigned to a link in \( G' \) is the same as the \( (d-1) \)-dimensional interval associated with that edge in the \( \{1,d-1\} \)-MLIRS defined on \( G' \). The \( d \)th coordinate of all intervals is set to be \([1..M]\). Any \( (d-1) \)-dimensional interval in \( G' \) that does not contain \( v_1 \) or \( v_2 \) will still contain the same set of vertices and any interval containing \( v_1 \) (respectively \( v_2 \)) will also contain all the vertices in \( C_1 \) (\( C_2 \)). For example the two dimensional interval \( I \), shown in Figure 7 (a), contains \( v_1 \), so the new three-
dimensional interval $I'$ contains all the vertices in $C_1$. Since $I$ does not contain $v_2$, $I'$ does not contain any of the vertices in $C_2$.

For the intervals associated with the links in $C_1$ or $C_2$, the first $d - 1$ coordinates are set to $[1..n]$. To set the $d$th coordinate of each interval we will use the previously mentioned $(1,1)$-MLIRS. Let us assume that in the $(1,1)$-MLIRS defined on $C_1$ the interval assigned to a link $e$ is $I_e = [a..b]$. If $I_e$ does not contain $u_1$, the $d$th coordinate of the newly assigned $d$-dimensional interval will be $[a + s..b + s]$ (we shift the $d$th coordinate by $s$ units because we have already shifted the vertices in this dimension). If $I_e$ contains $u_1$, i.e. $I_e = [1..b]$ for some $b$, the $d$th coordinate of the newly assigned interval will be $I_e = [1..b + s]$. This means that any 1-dimensional interval defined in $C_1$ will be transformed into a $d$-dimensional interval containing the same set of vertices in $C_1$ and if it contains $u_1$, it will also contain all the vertices in $G'$ and $C_2$. The interval $I$ depicted in Figure 7 (b) contains $u_1$, so the new interval $I'$ contains the set of vertices in $C_1$ that where in $I$ and also all the vertices in $C_2$ and $G'$. We will analogously assign intervals to the links in $C_2$.

The only remaining labels to update are labels of the links $(v_1, u_1), (u_1, v_1), (v_2, u_2)$ and $(u_2, v_2)$. The first $d - 1$ coordinates of intervals associated with $(v_1, u_1), (u_1, v_1), (v_2, u_2)$ and $(u_2, v_2)$ are set to $[1..n]$ and the $d$th coordinates will respectively be $[s + 1..n], [1..s], [1..s - 1]$ and $[s..n]$.

**Correctness:**

Now, let us consider a message originating from vertex $w_s$ and with destination $w_t$. If both $w_s$ and $w_t$ are in $C_1$ (similarly $C_2$ or $G'$) one can easily check that the newly defined $(1,d)$-MLIRS will route the messages on the same path as the $(1,1)$-MLIRS defined on $C_1$ ($C_2$ or the $(1,d - 1)$-MLIRS defined on $G'$). This is because if we just considering the set of vertices in $C_1$ ($C_2$ or $G'$) each interval assigned to a link contains the same set of vertices as it contained before expanding the labels to $d$ dimensions. If $w_s$ is in $C_1$ and $w_t$ in $G'$, the message must go through the link $(u_1, v_1)$ because this is the only link connecting $C_1$ to $G'$. The intervals in $C_1$ which contain $w_t$ are exactly the intervals containing $u_1$. Therefore, this message will be forwarded through the same links as the links through which a message towards $u_1$ would be forwarded. When the message reaches $u_1$, the bridge $(u_1, v_1)$ forwards the message to $v_1$, because the interval assigned to $(u_1, v_1)$ contains all the vertices in $G'$ and $C_2$. The rest of the routing will be the same as the $(1,d - 1)$-MLIRS defined on $G'$.

We can show that if there is a message in node $x$ ($x = u_2, v_1$ or $v_2$) which is supposed to be forwarded the bridge connected to $x$, say $e_s$, ($e_s = (u_2, v_2), (v_1, u_1)$ or $(v_2, u_2)$ respectively), will be sent to the other end of $e_s$. Verifying the cases in which $w_s$ is in $C_2$ or $G'$ is similar. Hence, any message originating at any vertex and going to an arbitrary destination will eventually reach the destination, and the $(1,d)$-MLIRS routes messages on $G$ properly.

We now have shown that if a graph is not in the class of $(2d + 1)$-windmill
graphs it has a \(\langle 1, d\rangle\)-MLIRS. Lemma 1 shows that no graph in this class can support a \(\langle 1, d\rangle\)-MLIRS. Combining these two results completes the proof of the theorem.

Since for each \(d > 1\), we have a \((2d + 1)\)-windmill graph which is not a \((2d + 3)\)-windmill graph (for example the \(Y_{2d+1}\) graph), we can state the following corollary:

**Corollary 1** The class of graphs supporting \(\langle 1, d\rangle\)-MLIRS is a strict subset of the class of graphs supporting \(\langle 1, d + 1\rangle\)-MLIRS.

In other words, increasing the number of dimensions increases the power of the routing scheme.

4 Characterization of networks supporting \(\langle 1, d\rangle\)-MSLIRS

In this section we will give a characterization of the class of graphs supporting \(\langle 1, d\rangle\)-MSLIRS. We will give some new definitions and will show that with slight changes in some steps in proofs, we can use the same ideas used to characterize the class of graphs supporting \(\langle 1, d\rangle\)-MLIRS.

In proving the Lemma 1, we needed to have at least two vertices in each arm of a \((2d + 1)\)-windmill graph. Otherwise, if the arm which did not have any vertex in the boundary set, say \(A_i\), had just one vertex, say \(x\), the interval assigned to the edge connecting \(A_i\) to \(R\) could contain \(x\) and this was not a contradiction. On the other hand, if the intervals assigned to the links are supposed to be strict, we could prove a similar lemma, even if we had an arm having just one vertex. This is the main difference between the proofs of this section and the previous one. More formally, let us start with a new definition.

**Definition 4** A weak \(k\)-windmill graph is a connected graph \(G\) with \(k + 1\) connected components \(A_1, A_2, \ldots, A_k\) (arms) and \(R\) (center) such that:

(i) there is no edge in \(G\) connecting \(A_i\) to \(A_j\) for \(1 \leq i, j \leq k\) and \(i \neq j\);

(ii) each component \(A_i, 1 \leq i \leq k\) is connected with \(R\) by exactly one bridge (Figure 8).

As mentioned above, if the IRS is strict, then with even one vertex in each arm the proof of Lemma 1 will still be valid, because a vertex which is not in the boundary set is contained in an edge connected to it. Therefore, any weak \((2d + 1)\)-windmill graph does not have a \(\langle 1, d\rangle\)-MSLIRS. We can also verify, with the same argument as the proof of Lemma 4, that removing any perfect chain from
Figure 8: A weak 5-windmill graph.

a graph $G$ which is not a weak $k$-windmill graph will produce a graph which is not a weak $(k-1)$-windmill graph.

The only remaining step is to show that the induction basis and step are also valid in constructing a $\langle 1, d \rangle$-MSLIRS for any graph that is not a weak $(2d + 1)$-windmill graph. We already know that any graph which is not weak 3-windmill graph (a weak lithium graph as defined in [5]) has a $\langle 1, 1 \rangle$-MSLIRS, so the induction basis is true. Since we have lemmas similar to Lemmas 3 and 4 one can verify that a similar induction step still works here. This give us the complete characterization of graph supporting $\langle 1, d \rangle$-MSLIRS as follows:

**Theorem 2** A graph $G$ has a $\langle 1, d \rangle$-MSLIRS if and only if it is not a weak $(2d + 1)$-windmill graph.

**Corollary 2** The class of graphs supporting $\langle 1, d \rangle$-MSLIRS is a strict subset of the class of graphs supporting $\langle 1, d + 1 \rangle$-MSLIRS.

## 5 Conclusions and open problems

In this paper we completely characterized the class of networks supporting $\langle 1, d \rangle$-MLIRS and the class of networks supporting $\langle 1, d \rangle$-MSLIRS. We showed that increasing the number of dimensions makes the routing scheme more powerful. One natural extension to this problem is to characterize the networks having a $\langle 1, d \rangle$-MLIRS or $\langle 1, d \rangle$-MSLIRS when the network has weighted links with dynamic costs. If the routing paths are supposed to be shortest paths, and we can relabel the edges after each change in the cost of links, there is a complete characterization for $\langle 1, d \rangle$-MSLIRS [8]. If the intervals are the same for any costs of links, the characterization problem is open even except for the 1-dimensional case.
There is a partial characterization for the class of networks supporting optimum \textit{LIRS} in 1-dimension \cite{10}. Finally, one can consider the problem of finding bounds on the length of routing paths for each of these classes.

6 Acknowledgment

I would like express my sincere gratitude to my supervisor, Prof. Naomi Nishimura, for her thoughtful comments, guidance and support.

References


Yashar Ganjali received a B.Sc. degree in Computer Engineering from the Computer Engineering Department, Sharif University of Technology, Tehran, Iran. Currently, he is a M.Sc. student in the Department of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1. E-mail: yganjali@math.uwaterloo.ca
A Characterization of Classes of Graphs Recognizable by Local Computations with Initial Knowledge

(Extended abstract)

E. GODARD
LaBRI, Université Bordeaux I, France

Y. MÉTIVIER
LaBRI, Université Bordeaux I - ENSEIRB, France

Abstract
This paper investigates the power of local computations on graphs considering the classical recognition problem. The recognition problem asks for computing some topology information on a network of anonymous processes, assuming some initial knowledge about the underlying graph.

We have introduced in [GMM00] a definition of recognition with initial knowledge by means of local relations and we have given a necessary condition for a class of graphs to be recognizable. In this paper we prove that this condition is sufficient. Some applications of this complete characterization are presented.

Keywords
initial knowledge, local computations, recognizable classes of graphs

1 Introduction

Local computations on graphs which are described by graph rewriting systems [LMS99] are an adequate model for distributed computing. Rewriting systems provide a general tool for describing and analyzing distributed algorithms and for proving their correctness. In this paper we consider local computations having some initial knowledge of the network.

We focus on the classical recognition problem. We consider graphs which are uniformly labelled by some initial label (which may encode some knowledge on the graph) and the presence or the absence of certain final labels determine whether the graph is accepted or not. The recognition problem asks for graph
properties like acyclicity, planarity etc. Several basic properties like regularity, completeness, or acyclicity can be recognized by local computations without initial knowledge. On the other hand, we cannot determine whether a graph is planar by local computations, provided that the given graph is labelled in an uniform way without initial knowledge [LMZ95, CM94]. However, the presence of a distinguished vertex allows to gather information. In particular, it has been shown that it is possible to detect a given minor in a graph with a distinguished vertex [BM98], hence also to determine whether the graph is planar. A natural question is whether some additional information encoded in the initial uniform labelling of a graph can help for deciding for example planarity.

The classical proof techniques used for showing the non-existence of local solutions are based on coverings [Ang80, FLM86], which is a notion known from algebraic topology. Coverings have been also used for simulation [BL86]. A graph \( G \) is a covering of a graph \( H \) if there is a surjective morphism from \( G \) to \( H \) which is locally bijective. The general idea is as follows. If \( G \) and \( H \) are two graphs such that \( G \) covers \( H \), then every local computation on \( H \) induces a local computation on \( G \). As a consequence, every class of graphs which is recognizable by local computations (without any knowledge of the graph) must be closed under coverings. Using this fact it has been proved that the class of planar graphs is not recognizable by local computations [LMZ95]. More generally, it has been shown that up to some few exceptions, every minor-closed class of graphs (that is, a class of graphs characterized by a finite set of forbidden minors by the Graph Minor Theorem) is not closed under coverings, implying that it is not recognizable by local computations [CM94]. In particular, one cannot decide by local computations whether a given graph is included as a minor in an arbitrary graph.

A graph \( G \) is called covering-minimal if every covering from \( G \) to some \( H \) is a bijection. The class of covering-minimal graphs plays an important role in the study of local computations: knowing the size, there exists an election algorithm for the class of covering-minimal graphs [Maz97]. It is easy to verify using prime rings, that the property of being covering-minimal is not recognizable without any initial knowledge of the graph. Using [Maz97] we note that this property is recognizable if we have as initial knowledge the size of the graph. Having an odd number of vertices or having exactly one vertex with a certain label are other examples of properties which are not recognizable without initial knowledge, but become recognizable if the graph size is known. Thus, recognizability under the assumption that the size is known to the algorithm is significantly more powerful than recognizability without initial knowledge.

To study recognizable class of graphs with some additional knowledge, we cannot apply the covering argument directly. In [GMM00] we have introduced a new construction. Let \( t \) be a function which codes some knowledge on graphs, we consider that two graphs \( G \) and \( G' \) are in relation if \( t(G) = t(G') \) and there exists a graph \( H \) such that \( G \) and \( G' \) are coverings of \( H \). Let ~ denote the transitive closure of this relation. We have proved in [GMM00] that a recognizable class of graphs
must be closed under this relation. In this paper we prove the converse and thus we obtain a characterization (Theorem 2) of classes of graphs recognizable with initial knowledge by local computations: a class of graphs $\mathcal{F}$ is locally recognizable with some additional knowledge $\iota$ if and only if it is recursive and closed with respect to the relation $\sim$. We give various applications of this theorem. In particular, we deduce immediately that classes of graphs that are recognizable are stable by union, intersection and complement.

**Related work.** Among models related to our model there are local computation systems as defined by Rosenstiehl et al. [RFH72], Angluin [Ang80] and Yamashita and Kameda [YK96]. In [RFH72] a synchronous model is considered, where vertices represent (identical) deterministic finite automata. The basic computation step is to compute the next state of each processor according to its state and the states of its neighbours. In [Ang80] an asynchronous model is considered. A basic computation step means that two adjacent vertices exchange their labels and then compute new ones. In [YK96] an asynchronous model is studied where a basic computation step means that a processor either changes its state and sends a message or it receives a message.

Please, note that this paper is an extended abstract of [GM01].

## 2 Basic Notions and Notation

### 2.1 Labelled Graphs

The notation used here is essentially standard [Ros00]. We only consider finite, undirected and connected graphs without multiple edges and self-loops.

Let $k$ be an integer. We say that a graph $G$ is a $k$-covering of a graph $H$ if there exists a surjective homomorphism $\gamma$ from $G$ onto $H$ such that for every vertex $v$ of $V(G)$ the restriction of $\gamma$ to $B_G(v,k)$ is a bijection onto $B_H(\gamma(v),k)$. The covering is proper if $G$ and $H$ are not isomorphic. It is called connected if $G$ (and thus also $H$) is connected. A graph $G$ is called $k$-covering-minimal if every $k$-covering from $G$ to some $H$ is a bijection. Moreover, for any $k < k'$ a $k'$-covering is also a $k$-covering. Note that a 1-covering is exactly a covering in the classical sense.

Throughout the paper we will consider only connected graphs where vertices and edges are labelled with labels from a possibly infinite alphabet $L$. A graph labelled over $L$ will be denoted by $(G, \lambda)$, where $G$ is a graph and $\lambda: V(G) \cup E(G) \rightarrow L$ is the labelling function (in most cases we use only a vertex-labelling function). The graph $G$ is called the underlying graph and the mapping $\lambda$ is a labelling of $G$. The class of labelled graphs over some fixed alphabet $L$ will be denoted by $\mathcal{G}$. 

Let $(G, \lambda)$ and $(G', \lambda')$ be two labelled graphs. Then $(G, \lambda)$ is a subgraph of $(G', \lambda')$, denoted by $(G, \lambda) \subseteq (G', \lambda')$, if $G$ is a subgraph of $G'$ and $\lambda$ is the restriction of the labelling $\lambda'$ to $V(G) \cup E(G)$. 

A mapping $\varphi: V(G) \cup E(G) \rightarrow V(G') \cup E(G')$ is a homomorphism from $(G, \lambda)$ to $(G', \lambda')$ if $\varphi$ is a graph homomorphism from $G$ to $G'$ which preserves the labelling, i.e. such that $\lambda'(\varphi(x)) = \lambda(x)$ holds for every $x \in V(G) \cup E(G)$. The mapping $\varphi$ is an isomorphism if it is bijective.

We extend the notion of $k$-covering to labelled graphs in an obvious way. The labelled graph $(H, \lambda')$ is covered by $(G, \lambda)$ via $\gamma$, if $\gamma$ is a homomorphism from $(G, \lambda)$ to $(H, \lambda')$ whose restriction to $B_G(v, k)$ is an isomorphism from $B_G(v, k)$ to $B_H(\gamma(v), k')$.

An occurrence of $(G, \lambda)$ in $(G', \lambda')$ is an isomorphism $\varphi$ between $(G, \lambda)$ and a subgraph $(H, \eta)$ of $(G', \lambda')$.

### 2.2 Local Computations

Local computations as considered here can be described in the following general framework. Let $\mathcal{G}$ be the class of $L$-labelled graphs and let $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$ be a binary relation on $\mathcal{G}$. Then $\mathcal{R}$ will denote a graph rewriting relation. We assume that $\mathcal{R}$ is closed by isomorphism, i.e., whenever $(G, \lambda) \mathcal{R} (G', \lambda')$ if $(G_1, \lambda_1) \simeq (G, \lambda)$ then $(G_1, \lambda_1) \mathcal{R} (G_2, \lambda_2)$ for some labelled graph $(G_2, \lambda_2)$ isomorphic to $(G', \lambda')$. In the remainder of this paper $\mathcal{R}^*$ stands for the reflexive and transitive closure of $\mathcal{R}$. The labelled graph $(G, \lambda)$ is $\mathcal{R}$-irreducible if there is no $(G', \lambda')$ such that $(G, \lambda) \mathcal{R} (G', \lambda')$. Let $(G, \lambda) \in \mathcal{G}$, then $\text{Irred}_k((G, \lambda))$ denotes the set of $\mathcal{R}$-irreducible graphs (or just irreducible if $\mathcal{R}$ is fixed) which can be obtained from $(G, \lambda)$ using $\mathcal{R}$. The relation $\mathcal{R}$ is noetherian if there is no infinite chain $(G_1, \lambda_1) \mathcal{R} (G_2, \lambda_2) \mathcal{R} \ldots$. 

**Definition 1** Let $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$ be a graph rewriting relation and let $k > 0$ be an integer.

1. $\mathcal{R}$ is a relabelling relation if whenever two labelled graphs are in relation then the underlying graphs are equal (we say equal, not only isomorphic), i.e. $$(G, \lambda) \mathcal{R} (H, \lambda') \implies G = H.$$ 

2. $\mathcal{R}$ is $k$-local if only labels of a ball of radius $k$ may be changed by $\mathcal{R}$, i.e., $(G, \lambda) \mathcal{R} (G', \lambda')$ implies that there exists a vertex $v \in V(G)$ such that $$\lambda(x) = \lambda'(x) \text{ for every } x \not\in V(B_G(v, k)) \cup E(B_G(v, k)).$$

The relation $\mathcal{R}$ is local if it is $k$-local for some $k > 0$.

The next definition states that a local relabelling relation $\mathcal{R}$ is $k$-locally generated if its restriction on centered balls of radius $k$ determines its computation on any graph.

**Definition 2** Let $\mathcal{R}$ be a relabelling relation and $k > 0$ be an integer. Then $\mathcal{R}$ is $k$-locally generated if the following is satisfied: For every labelled graphs $(G, \lambda)$,
\[(G, \lambda'), (H, \eta), (H, \eta')\) and every vertices \(v \in V(G), w \in V(H)\) such that the balls \(B_G(v, k)\) and \(B_H(w, k)\) are isomorphic via \(\varphi: V(B_G(v, k)) \rightarrow V(B_H(w, k))\) and \(\varphi(v) = w\), the following three conditions

1. \(\lambda(x) = \eta(\varphi(x))\) and \(\lambda'(x) = \eta'(\varphi(x))\) for all \(x \in V(B_G(v, k)) \cup E(B_G(v, k))\)

2. \(\lambda(x) = \lambda'(x)\), for all \(x \notin V(B_G(v, k)) \cup E(B_G(v, k))\)

3. \(\eta(x) = \eta'(x)\), for all \(x \notin V(B_H(w, k)) \cup E(B_H(w, k))\)

imply that \((G, \lambda) R_1(G, \lambda')\) if and only if \((H, \eta) R_2(H, \eta')\).

We give now a fundamental lemma which connects \(k\)-coverings and \(k\)-locally generated relabelling relations. It states that whenever \(G\) is a \(k\)-covering of \(G'\), every \(k\)-local computation in \(G'\) can be lifted to a \(k\)-local computation in \(G\) which is compatible with the \(k\)-covering relation. This is expressed in the following lemma:

**Lemma 1** Let \(R\) be a \(k\)-locally generated relabelling relation and let \((G, \lambda_1)\) be a \(k\)-covering of \((G', \lambda'_1)\) via \(\gamma\). Moreover, let \((G', \lambda'_1) R_1(G', \lambda'_2)\). Then a labelling \(\lambda_2\) of \(G\) exists such that \((G, \lambda_1) R_2(G, \lambda_2)\) and \((G, \lambda_2)\) is a \(k\)-covering of \((G', \lambda'_2)\).

### 2.3 Distributed Computations of Local Computations

The notion of relabelling sequence defined above, obviously corresponds to a notion of sequential computation. Let us also note that a \(k\)-locally generated relabelling relation also allows parallel rewritings, since non-overlapping \(k\)-balls may be relabelled independently. Thus we can define a distributed way of computing by saying that two consecutive relabelling steps concerning non-overlapping \(k\)-balls may be applied in any order. We say that such relabelling steps commute and they may be applied concurrently. More generally, any two relabelling sequences such that the latter one may be obtained from the former one by a succession of such commutations lead to the same resulting labelled graph. Hence, our notion of relabelling sequence may be regarded as a serialization \([\text{Maz87}]\) of some distributed computation. This model is clearly asynchronous: several relabelling steps may be done at the same time but we do not require that all of them have to be performed. In the sequel we will essentially deal with sequential relabelling sequences but the reader should keep in mind that such sequences may be done in a distributed way.

### 3 Graph Recognizers and Initial Knowledge

Let \(L\) be a set of labels. The problem addressed in this section can be informally described as follows. Let \(\mathcal{F}\) some class of (labelled) graphs. We will say that \(\mathcal{F}\)
can be locally recognized if there exists some graph relabelling system (or, more generally, some locally generated graph relabelling relation) such that starting from any uniformly labelled graph \((G, \lambda_0)\) some final labelling can be reached that allows to decide whether \(G\) belongs to the class \(\mathcal{F}\) or not.

**Definition 3** A labelled graph recognizer is a pair \((\mathcal{R}, \mathcal{K})\) where \(\mathcal{R}\) is a graph relabelling relation and \(\mathcal{K}\) is a class of labelled graphs.

A graph \((G, \lambda)\) is recognized if \(\text{Irred}_\mathcal{R}(G, \lambda) \cap \mathcal{K} \neq \emptyset\).

We are interested in recognizing graphs which have a certain initial knowledge encoded in the initial labelling. Let \(G\) be a graph and \(\alpha\) a label in \(L\). Then \(\Lambda_\alpha\) is the uniform labelling on \(G\) with \(\alpha\), that is every vertex is labelled by \(\alpha\).

**Definition 4** A graph recognizer with initial knowledge is a triple \((\mathcal{R}, \mathcal{K}, \iota)\) where \((\mathcal{R}, \mathcal{K})\) is a labelled graph recognizer, and \(\iota\) is a function which associates with each graph \(G\) a label \(\iota(G)\) in \(L\). The set of graphs recognized by \((\mathcal{R}, \mathcal{K}, \iota)\) is given as \(\{G \mid (G, \Lambda_{\iota(G)}) \text{ is recognized by } (\mathcal{R}, \mathcal{K})\}\).

A recognizer \((\mathcal{R}, \mathcal{K}, \iota)\) is said to be deterministic if, restricted to inputs \((G, \Lambda_{\iota(G)})\), we have the following two properties:

- \(\mathcal{R}\) is noetherian.
- Either \(\text{Irred}(G, \Lambda_{\iota(G)}) \cap \mathcal{K} = \emptyset\) or \(\text{Irred}(G, \Lambda_{\iota(G)}) \subseteq \mathcal{K}\).

We can now define recognizable classes of graphs.

**Definition 5** A class \(\mathcal{F}\) of graphs is said to be (deterministically) recognizable with initial knowledge \(\iota\) if there exists a locally generated (deterministic) graph recognizer \((\mathcal{R}, \mathcal{K}, \iota)\) recognizing exactly \(\mathcal{F}\).

In this paper, we deal with graph recognizers where the relation \(\mathcal{R}\) is locally generated. Moreover, the set \(\mathcal{K}\) is defined by means of a so-called final condition, which represents a logical formula inductively defined as follows: (i) for every label \(\ell \in L\), \(\ell\) is a formula and (ii) if \(\phi\) and \(\psi\) are formulas then so are \(\neg \phi\), \(\phi \lor \psi\) and \(\phi \land \psi\). Now, for \(\ell \in L\), a labelled graph satisfies the formula \(\ell\) if it contains at least one \(\ell\)-labelled vertex or edge, and by induction, it satisfies \(\phi \lor \psi\) if it satisfies \(\phi\) or \(\psi\) and so on in the usual way. Thus, such final conditions work only on the set of labels appearing in a given final labelling, or in other words, they allow to verify the presence or the absence of some specific labels but not to count the number of labels. We will denote by \(\mathcal{K}(\phi)\) the set of labelled graphs which satisfy the formula \(\phi\) (some examples are given in [LMZ95]).

A simple example of recognition system is given below.

**Example 1** Tree recognition

The following relabelling system recognizes trees.

The set of labels is \(\{\varepsilon, F, T\}\). As there is no initial information, \(\varepsilon\) is the initial label on all vertices.
First rule: If a node has exactly one $\varepsilon$-labelled neighbour then it changes its label to $F$.

Second Rule: If a node has no $\varepsilon$-labelled neighbour then it changes its label to $T$.

The final condition is the presence of the label $T$.

Remark: In $\iota$, we can encode some information about the underlying graphs such as an upper bound on the number of vertices, a tight bound (that is a bound $b$ such that for all graphs $G$, $|V(G)| \leq b(G) < 2|V(G)|$), the exact number of vertices or even the topology.

Remark: The knowledge of the topology does not solve all problems; in particular it does not enable to solve the election problem [GMM00]. For the recognition problem it remains to compute the correspondence between vertices of the concrete graph and vertices of the abstract graph given by the initial knowledge.

We give now a simple relation between graph recognizers and $k$-coverings from which we obtain new results on graph recognizers working on graphs with some initial knowledge.

For $k > 0$, we define a relation $\sigma_k^1$ by letting $G \sigma_k^1 G'$ if:

- $t(G) = t(G')$
- There exists a graph $H$ such that $G$ and $G'$ are $k$-coverings of $H$.

Let $\sim_k^1$ denote the reflexive, transitive closure of $\sigma_k^1$. A class of graphs $\mathcal{F}$ will be said to be closed under $\sim_k^1$ if for any graphs $G$ and $G'$ such that $G \sigma_k^1 G'$, $G$ is in $\mathcal{F}$ if and only if $G'$ is in $\mathcal{F}$. We obtain the following necessary condition for recognizability:

**Proposition 1** [GMM00] Let $\mathcal{F}$ be a class of graphs which is deterministically recognizable using the initial knowledge $\iota$. Then $\mathcal{F}$ is closed under $\sim_k^1$.

## 4 Enumeration and Recognition

To prove the converse of Proposition 1, we describe an algorithm that recognizes a given $\sim_k^1$-closed, recursive class of graphs, using the initial knowledge $\iota$. Our results will be shown using the algorithm given by Mazurkiewicz in [Maz97]. It is worth noting that despite its simplicity, this algorithm provides all the underlying graph information that a distributed algorithm can be required to compute.

### 4.1 Mazurkiewicz Algorithm: $\mathcal{M}_k$

We give here a general description of the algorithm $\mathcal{M}_k$. Initially all vertices have the same label. Every vertex attempts to get its own name, which shall be an integer between 1 and $|V|$. A vertex chooses a name and broadcasts it together with its
labelled neighbourhood all over the network. If a vertex \( u \) discovers the existence of another vertex \( v \) with the same name, then it compares its \textit{neighbourhood view}, i.e., its labelled \( k \)-ball, with the neighbourhood view of its rival \( v \). If the neighbourhood view of \( v \) is “stronger”, then \( u \) chooses another name. Each new name is broadcasted again over the network. At the end of the computation it is not guaranteed that every node has a unique name, unless the graph is covering-minimal.

However, all nodes with the same name will have the same \textit{neighbourhood view}, i.e., isomorphic labelled \( k \)-balls.

A total order \( \prec \) is defined on the set of neighbourhood view \( \mathcal{B} \). A vertex \( v \) is labelled by a tuple of the form \((n(v), N(v), M(v), d(v))\) representing the following information during the computation:

- \( n(v) \in \mathbb{N} \) is the name of the vertex \( v \),
- \( N(v) = (B, \lambda) \in \mathcal{L} \) is the neighbourhood view, i.e., \( B \) is a copy of \( B(v, k) \) and for all \( u \in V(B), \lambda(u) = n(u) \).
- \( M(v) \subseteq \mathbb{N} \times \mathcal{B} \) is the mailbox of \( v \) and contains the information received at this step of the computation,
- \( f(v) \in \{\text{UpToDate}, \text{DATED}\} \) is a flag that will be set to \text{DATED} when the vertex has to update its neighbourhood information.

The initial labelling of all vertices is \((0, \emptyset, \emptyset, \text{UpToDate})\).

The rules are described below for a given centered \( k \)-ball \( B = B(v_0, k) \) with center \( v_0 \). The vertices \( v \) of \( B \) have labels \((n(v), N(v), M(v), f(v))\). The labels obtained after applying a rule are \((n'(v), N'(v), M'(v), f'(v))\). To make the rules easier to read, we omit labels that are left unchanged.

\begin{enumerate}
\item \textbf{Diffusion rule:}
Suppose that \( f(v) = \text{UpToDate} \) for every \( v \in B \). Moreover, the mailboxes of \( B \) are inconsistent, i.e., \( M(v_0) \neq M(v) \), for some \( v \in B \).

The diffusion rule yields: \( M'(v) = \bigcup_{w \in B} M(w) \), for all \( v \in B \).

\item \textbf{Renaming rule:}
Suppose that \( f(v) = \text{UpToDate} \) for every \( v \in B \) and \( M(v) = M(v_0) \) for all \( v \in B \). Moreover, either \( n(v_0) = 0 \) or there is some \((n(v_0), N_1) \in M(v_0)\) such that \( N(v_0) \prec N_1 \), (i.e., there is a node with the same name and a “stronger” neighbourhood view). The renaming rule yields:

\begin{enumerate}
\item \( n'(v_0) = 1 + \max\{n \in \mathbb{N} \mid (n, N) \in M(v_0) \text{ for some } N \in \mathcal{L}\}\)
\item The neighbourhood view \( N'(v_0) \) is obtained from \( N(v_0) \) by updating the name of \( v_0 \).
\end{enumerate}
\end{enumerate}
(c) The mailbox contents changes to \( M'(v_0) = M(v_0) \cup \{(n'(v_0), N'(v_0))\} \).

(d) For all \( v \in B(v_0, k) \setminus \{v_0\} \), let \( f'(v) = \text{DATED} \).

(3) **Updating rule:**

Suppose that \( f(v_0) = \text{DATED} \). Then \( v_0 \) updates its neighbourhood view by letting \( N'(v_0) = (B(v_0, k), \lambda) \), where \( \lambda \) is given by \( \lambda(v) = n(v) \). The new value \( f'(v_0) \) of the flag is \( \text{UpToDATE} \).

Note that for \( k = 1 \) the updating rule is not needed, [Maz97]. The reason is that a neighbourhood of radius 1 is just a set, hence the neighbourhood views of neighbours can be updated within the renaming rule.

### 4.2 Properties of Mazurkiewicz Algorithm

Let \( G \) be a graph, then the labelling function obtained after a run \( \rho \) of Mazurkiewicz algorithm is noted \( \Pi_\rho \). If \( v \) is a vertex of \( G \), the 4-tuple \( \Pi_\rho(v) \) associated with \( v \) is denoted \((n_\rho(v), N_\rho(v), M_\rho(v), f_\rho(v))\). We obtain the following extension of Mazurkiewicz algorithm:

**Theorem 1** A run of Mazurkiewicz Enumeration Algorithm on a connected graph \( G \) terminates and yields a final labelling \( \Pi_\rho \) verifying the following conditions for all vertices \( v, v' \) of \( G \):

1. Let \( m \) be the maximal name in \( G \), \( m = \max_{v \in V(G)} n_\rho(v) \). Then for every \( 1 \leq l \leq m \) there is some \( v \in V(G) \) with \( l = n_\rho(v) \).

2. \( M_\rho(v) = M_\rho(v') \).

3. \((n_\rho(v), N_\rho(v)) \in M_\rho(v')\).

4. \( f_\rho(v) = \text{UpToDATE} \).

5. Let \((n, N) \in M_\rho(v')\). Then \( n_\rho(v) = n \) and \( N_\rho(v) = N \) for some vertex \( v \) if and only if there is no pair \((n, N') \in M_\rho(v')\) with \( N \prec N' \).

6. \( n_\rho \) induces a \( k \)-locally bijective labelling of \( G \).

We interpret the final labelling \( \Pi_\rho \) as a graph that each vertex could compute. For a mailbox \( M \), we define

\[
F(M) = \{(n, N) \in M \mid N \prec N \text{ for all } (n, N') \in M\}.
\]

For a given \( M \) we define the graph \( G_M \) as the following graph:

\[
V(G_M) = \{n \mid \exists N, (n, N) \in F(M)\}
\]

\[
E(G_M) = \{\{(n, n') \mid \exists N = (B, \lambda) \text{ such that } (n, N) \in F(M), \exists u, u' \in V(B), \\
\lambda(u) = n, \lambda(u') = n' \text{ and } \{u, u'\} \in E(B)\}\}
\]
Let $\rho$ be a run of $\mathcal{M}_k$, then $G_\rho = G_{M_k(\rho)}$ does not depend on $u$, as shown by Theorem 1, we define $G_\rho = G_{M_k(\rho)}$.

**Proposition 2** Let $G$ be a graph.

1. For all runs $\rho$ of $\mathcal{M}_k$, $G$ is a $k$–covering of $G_\rho$.
2. For all $H$ such that $G$ is a $k$–covering of $H$, there exists a run $\rho$ such that $H \simeq G_\rho$.

### 4.3 Recognizing $\sim_k^1$ – closed Graph Classes

In this section, we sketch a distributed algorithm for recognizing a given recursive $\sim_k^1$–closed class of graphs $\mathcal{F}$. Throughout the section we suppose that $\mathcal{F}$ is recursive and closed under $\sim_k^1$ without further mentioning it. We first need some notation. Let

$$
\mathcal{F}_k^1 = \{ (H, t(G)) \mid G \in \mathcal{F} \text{ and } G \text{ is a } k\text{-covering of } H \}.
$$

We have this simple and fundamental lemma:

**Lemma 2** Let $\mathcal{F}$ be a $\sim_k^1$–closed class of graphs. Let $G$ be a graph. Then for any run $\rho$ of $\mathcal{M}_k$:

$$
G \in \mathcal{F} \text{ if and only if } (G_\rho, t(G)) \in \mathcal{F}_k^1.
$$

We also remark that:

**Lemma 3** For any $(H, \alpha)$, we have

1. $(H, \alpha) \in \mathcal{F}_k^1 \iff \exists K \in \mathcal{F} \text{ satisfying } K \text{ is a } k\text{-covering of } H \text{ and } t(K) = \alpha.$
2. $(H, \alpha) \notin \mathcal{F}_k^1 \iff \exists K \notin \mathcal{F} \text{ satisfying } K \text{ is a } k\text{-covering of } H \text{ and } t(K) = \alpha.$

We add some information to the 4–tuple computed by Mazurkiewicz algorithm. More precisely, we consider on each node $v$ two boolean variables, denoted $b(v)$ and $r(v)$ whose initial values are false. When a node $v_0$ is relabelled by Mazurkiewicz algorithm, $b(v)$ and $r(v)$ become false. If no rule of Mazurkiewicz algorithm may be applied on the node $v_0$ then $b(v_0)$ becomes true if all nodes of the ball centered on $v_0$ can reconstruct the same graph. If $b(v)$ is true then the node run the algorithm of test described below on $(G_\rho, t(G))$ setting the result to $r(v)$.

We enumerate all the graph $K$. For each one we test if $K$ is a $k$–covering of $G_\rho$, and $t(K) = t(G)$. When we obtain a graph $K$ verifying these two conditions, we test if it belongs to $\mathcal{F}$ ($\mathcal{F}$ is recursive by hypothesis).

We also make some fairness assumptions on the execution on this algorithm, the execution is reseted whenever a $\mathcal{M}_k$ rule is applied, such that there is no forever enumeration. Those assumptions can be omitted provided some rather technical improvements not detailed here. Finally we have:
**Proposition 3** If for each node \( v \) of \( G \) \( b(v) = true \) then Mazurkiewicz algorithm has reached a terminal configuration. Conversely, when Mazurkiewicz algorithm has reached a terminal configuration then after a number of steps bounded by the size of \( G \) \( b(v) \) is true and remains true for all nodes of \( G \).

From Lemma 2 and Lemma 3, as \( \mathcal{F} \) is closed, we deduce whether \( G \) belongs to \( \mathcal{F} \) looking at the \( r(v) \) labels in the terminal configuration, i.e.:

**Proposition 4** When Mazurkiewicz algorithm has reached a terminal configuration, we have:

\[
G \in \mathcal{F} \iff \forall v \quad r(v) = true.
\]

As a synthesis of the previous results we obtain:

**Theorem 2** Let \( \mathcal{F} \) be a class of graphs and \( \tau \) an initial knowledge. The following statements are equivalent.

1. \( \mathcal{F} \) is \( k \)-locally recognizable with initial knowledge \( \tau \).
2. \( \mathcal{F} \) is closed under \( \sim_\tau \) and \( \mathcal{F} \) is a recursive set.

An interesting corollary is the stability by boolean operations.

**Corollary 1** Classes of graphs that are \( k \)-locally recognizable with initial knowledge \( \tau \) are stable for union, intersection, and complement.

### 5 Applications: Particular Cases of Initial Knowledge

The results of the previous section state that the \( \sim_\tau \)-equivalence classes are the atoms of recognizable classes of graphs. In this section we are investigating the properties of equivalence classes in some particular interesting cases.

#### 5.1 No a priori Knowledge

Leighton [Lei82] gives a decidable criterium for two graphs admitting a common covering. He uses the so-called degree partition of a graph \( G \), i.e., the partition of the vertices of \( G \) into the minimal number of blocks \( B_0, B_1, \ldots, B_{t-1} \) for which there are constants \( r_{i,j} \), \( 0 \leq i, j < t \), such that every vertex \( v \) in \( B_i \) is incident to \( r_{i,j} \) edges linking \( v \) to vertices in \( B_j \). The degree refinement of \( G \) is then the \( t \times t \) matrix \( R = (r_{i,j}) \). Two degree refinements \( R_1 \) and \( R_2 \) are considered to be the same if the two matrices are conjugated.

**Theorem 3** (Leighton, 1982) Given any two finite connected graphs \( G \) and \( H \), \( G \) and \( H \) share a common finite covering if and only if they have the same degree refinement.
Concerning our relation $\sim_\varepsilon^k$, we obtain for $k = 1$:

**Proposition 5** Let $G$ and $G'$ be two connected graphs. Then $G \sim_\varepsilon^1 G'$ if and only if $G$ and $G'$ have the same degree refinement.

**Remark:** We do not have such a characterization for $k > 1$.

As the degree refinement is computable on a graph we also obtain:

**Corollary 2** Let $G$ be a graph. The equivalence class of $G$ is $1-$recognizable with no initial knowledge.

In particular, $1-$recognizable graph classes with no initial knowledge can be seen as recursive sets of degree refinement.

## 5.2 Size Upbounds

We defined an upbound recognizable class to be a class that is recognizable by the relabelling system whatever the chosen upbounding function. It is then equivalent to say that an upbound recognizable class is a class that is closed for all $\sim^b_k$ relation. So, if we define $\sim^{\text{bound}}_k = \bigcup_b \sim^b_k$, where the union is taken over all upper bounds $b$ of the size of $G$, we have:

**Proposition 6** Let $\mathcal{F}$ be a class of graphs. $\mathcal{F}$ is upbound recognizable if and only if $\mathcal{F}$ is closed for $\sim^{\text{bound}}_k$

We have

**Lemma 4** For all $k \in \mathbb{N}$, $\sim_\varepsilon^k = \sim^{\text{bound}}_k$.

**Proof.** For every $k \in \mathbb{N}$, $\sim_\varepsilon^k \supset \sim^{\text{bound}}_k$ is obvious. For the other inclusion, suppose we have a graph $H$ that is a $k-$covering of $G$ and $G'$, then for an upper bound function such that $b(G) = b(G') = \max(|G|, |G'|)$, we have $G \sim^b_k G'$.

As an immediate corollary of the previous section, we obtain:

**Corollary 3** Let $\mathcal{F}$ be a class of graphs that is $k-$recognizable having a bound on the size. Then $\mathcal{F}$ is also $k-$recognizable with no initial information.

**Remark:** Having a bound on the number of vertices is the same as having no information in the recognition context. This , quite surprising, result is similar to one obtained in [YK96] (see Prop. 18). This is quite different in the termination detection context (see [MT00]).

## 5.3 Tight Upbound

We recall that a tight upperbound is a bound $b$ such that for all graphs $G$, $|V(G)| \leq b(G) < 2|V(G)|$. 
Proposition 7 The class of minimal graphs is recognizable knowing a tight upper bound.

Proof. Since a strict covering has at least two sheets, then the equivalence classes $[G]^\text{TightBound}_k$ are singletons. □

5.4 Knowing the Size

Remark: We can note that if $G$ and $H$ are both coverings of a graph $K$ and have the same number of vertices then they have the same number of edges and thus their cycle spaces have the same dimension.

That is to say that knowing the number of vertices is the same as knowing the number of edges.

5.5 Table

Let $k$ be any integer. We will summarize results in the following table where the YES and NO answer the question whether the listed family is recognizable or not by a deterministic $k$-generated graphs recognizer with the corresponding initial knowledge. The number within parenthesis refers to the proof below.

Proposition 8 We have:

<table>
<thead>
<tr>
<th>Family of graphs</th>
<th>No Info</th>
<th>Tight Bound</th>
<th>Number of vertices</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursive union of trees</td>
<td>YES (1)</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>k-Minimal graphs</td>
<td>NO (2)</td>
<td>YES (3)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Family of graphs such that $V(G)$ is even</td>
<td>–</td>
<td>NO (4)</td>
<td>YES (5)</td>
<td>–</td>
</tr>
<tr>
<td>Planar graphs</td>
<td>–</td>
<td>–</td>
<td>NO (6)</td>
<td>YES</td>
</tr>
<tr>
<td>Bipartite graphs</td>
<td>–</td>
<td>–</td>
<td>NO (7)</td>
<td>YES</td>
</tr>
</tbody>
</table>

Proof.

(1) By Theorem 2 and remarking that the $\sim_k$-class of a tree is a singleton.

(2) The family of $k$-minimal graphs is not closed for $k$-covering. Let us consider rings. The minimal rings are the ones that have a prime number of vertices.

(3) See Prop. 7

(4) if $p$ is an odd integer then consider the equivalence on the rings $R_{2p}G^\text{TightBound}_kR_{3p}$ via the ring $R_p$ and corollary 1.

(5) Trivial.
6 Conclusion

We present in this paper a characterization of classes of graphs that are recognizable by local computations with a given initial knowledge. Using a simple equivalence relation, we have established properties like the closure under boolean operations, and we have investigated the equivalence classes in some particular cases. Thus, we have partially answered the question of determining what initial knowledge is necessary in order to recognize some properties of the underlying graph by local computations. Further work will consist in a full description of the equivalence classes, especially with respect to quasi-coverings, which are used for termination detection [MT00].

References


**Emmanuel Godard** is with the LaBRI, Université Bordeaux I, 351, cours de la libération, 33405 Talence France. E-mail: godard@labri.u-bordeaux.fr

**Yves Métivier** is with the LaBRI, Université Bordeaux I - ENSEIRB, 351, cours de la libération, 33405 Talence France. E-mail: metivier@labri.u-bordeaux.fr
Modelling Links in Inclined LEO Satellite Networks*

PETER GVOZDJAK
Motorola Canada Ltd.

JOSEPH G. PETERS
Simon Fraser University, Canada

Abstract
We propose simple models for investigating communications in inclined low-earth-orbit (LEO) satellite networks. These networks connect satellites with intersatellite links. They differ from ground-based networks in several significant ways including the continuous motion of the satellites relative to each other. Our models are based on analytical and empirical investigations of link lengths and paths lengths in LEO networks. The models address two parameters of LEO networks—the selection of intersatellite links, and the modelling of link lengths.

Keywords
LEO Satellite Networks, Communication, Modelling

1 Introduction
In recent years, there have been several proposals to use networks of satellites in low earth orbits for communications. A big advantage of LEO satellite networks over Geosynchronous earth orbit (GEO) satellites is much smaller delay. LEO satellite orbits are less than 1500 km above the earth and typical earth-satellite-earth round trip delays are less than 10 msec. compared to more than 230 msec. for GEO satellites. However, the lower altitudes also introduce disadvantages including much shorter orbit periods, typically a few hours, and smaller footprints. This results in constant motion of the satellites with respect to the earth’s surface, and the need for more satellites to provide full coverage of the earth. LEO satellites can be used to supplement terrestrial networks by providing links between points on the surface that would be difficult or too expensive to connect with terrestrial links. A more ambitious approach is to replace large parts of terrestrial

* Supported by NSERC of Canada.
networks by using *intersatellite links* to interconnect a number of satellites into a *LEO satellite network*.

In this paper, we develop simple models for studying communication in LEO satellite networks. We start with a brief discussion of some general issues involved in LEO satellite network design in Section 2. In Sections 3 and 4, we determine the communication delays between two directly connected satellites and between arbitrary pairs of satellites which communicate along paths through several intermediate satellites. Our results show that minimizing the length of links between pairs of directly connected satellites usually minimizes end-to-end path lengths as well. We study the impact of various parameters on the communication delays with a particular focus on the interconnection pattern among satellites. We determine which interconnections minimize the delay between directly connected satellites and between arbitrary pairs of satellites. We show that a modified *toroidal mesh* interconnection topology called a *skewed torus* is a natural consequence of placing satellites in *inclined orbits* (which lie on planes that are at an angle to the plane that includes the earth’s axis of rotation).

In a LEO network, there are two types of intersatellite links: *intra-orbital links* between satellites in the same orbit, and *inter-orbital links* between satellites in adjacent orbits. When inclined orbits are used, the directions and lengths of inter-orbital links are constantly changing according to a complex formula. In Section 5, we propose two simple models that approximate the actual values and calculate the errors that they introduce on the average end-to-end delay. The numerical results show that the typical average error is under 5% for our *linear-cost* approximation, and under 20% for our *constant-cost* approximation. We have used the constant-cost approximation in [7] to investigate the *gossiping* communication pattern in which each satellite broadcasts a message to all other satellites.

## 2 LEO Satellite Networks

Perhaps the most important design issue for a LEO network is the choice of a *constellation*—the number of satellites, the number of orbits, and their *inclination*. The inclination of the orbits is either *polar* or *inclined*. *Polar orbits* lie on a plane that includes the earth’s axis of rotation, so satellites in polar orbits cross over both of the earth’s poles during each orbit. Two examples of LEO networks with polar constellations are described in [8] and [11]. Other topologies for polar constellations are discussed in [4, 5, 15]. Also see [2, 9, 14] for more general surveys. We will focus on *inclined constellations* which have orbits that lie on planes that include the center of the earth but not the poles. One advantage of inclined constellations compared to polar constellations is that more regular topologies are possible. In some cases, inclined constellations can achieve the same coverage with fewer satellites than polar constellations, particularly when the polar areas need not be covered [10]. One drawback of inclined constellations is higher
variability in the relative positions of satellites in different orbits and resulting difficulties with beam steering and Doppler shift [4].

All of the constellations that we will consider (and most of the proposed networks) are regular constellations that place the same number of uniformly spaced satellites in each orbit, position the orbits uniformly around the earth, and use the same angle of inclination for all orbits. Among other things, the choice of a constellation must take into account the desired coverage—the way the entire surface of the earth, or selected regions, are covered by satellite footprints. We will not discuss the interrelationships among constellations, coverage, and footprints in detail. See [13] for the details of the geometry of satellite coverage and [10] for an extensive discussion of coverage and the choice of constellations.

Once the constellation is chosen, the intersatellite topology is defined by selecting the intersatellite connections, or links, between satellites. We focus on fixed topologies in which each satellite always maintains links to the same set of satellites. Moreover, we assume that the topologies are symmetric, i.e., that the set of the connected satellites is defined in the same way for every satellite. The number of links per satellite can vary, and topologies ranging from 3 to 8 links per satellite have been proposed. In this paper, we assume four links per satellite, two intra-orbital links connecting it to its nearest neighbours in the same orbit, and one inter-orbital link to a satellite in each of the two neighbouring orbits. This allows for a reasonably dense and symmetric network. The satellites connected by intra-orbital links are in fixed relative positions and distances, so the links do not require steering of the transmitters and receivers. A natural choice seems to be connecting a satellite to its predecessor and successor in the same orbit. These connections appear in virtually all studies.

The focus of this paper is on the choice of inter-orbital links. There are several criteria that can be used when choosing the inter-orbital links including link length, average or maximum path length, and delay jitter. Other criteria are ease of handing off connections and steering of the beams. In this paper, we concentrate on the minimization of distances, or, equivalently, propagation delays between pairs of satellites.

Two satellites have the same phase if they are at the same latitude at any given time. If each satellite is connected to the satellites with the same phase in the two neighbouring orbits, then the resulting topology is a torus. However, as shown below, the intersatellite distances are usually minimized when the phases of connected satellites differ. This results in a modified toroidal mesh topology called a skewed torus [1]. An ordinary $n_h \times n_v$ torus has nodes labelled by integer pairs $(i,j), 0 \leq i \leq n_h - 1, 0 \leq j \leq n_v - 1$. The four neighbours of node $(i,j)$ are $((i \pm 1) \mod n_h, j)$ and $(i, (j \pm 1) \mod n_v)$. A k-skewed torus is formed by replacing the edges $[(n_h - 1, j), (0, j)], 0 \leq j \leq n_v - 1$, by the edges $[(n_h - 1, j), (0, (j + k) \mod n_v)], 0 \leq j \leq n_v - 1$. Conceptually, we cut the torus vertically between columns $n_h - 1$ and 0 and reconnect each node in column $n_h - 1$ to the node in column 0 that is $k$ rows up. An example is shown in Figure 1. Figure 2 shows
the interconnections for an inclined satellite network which has 4 orbits with the same phasing, 10 satellites per orbit, and skew 4. One proposal for an inclined constellation with 12 orbits, 6 satellites per orbit, and different phasing in the orbits is investigated in [12].

3 Intersatellite Link Length

To derive formulae for the length of intersatellite links, we will first introduce some terminology related to satellites and their orbits. (See Figure 3.) The inclination $\alpha_0$ of a satellite orbit is the angle between the orbital plane and the equatorial plane; note that all orbital planes intersect the center of the earth. The descending node of a satellite is the point at which the satellite crosses the equator in the direction from north to south. The angle $\phi$ between the descending node, the center of the earth and the satellite’s position will be called phase. We will consider a coordinate system centered at the center of the earth whose $xy$-plane coincides with the equatorial plane. To simplify the formulae, terms like latitude and longitude will refer to the fixed sphere associated with the coordinate system, not to a rotating earth, with 0° longitude in the direction of the $x$-axis.

The following lemma can be derived using elementary geometry.

**Lemma 1** Consider two satellites $S_1$ and $S_2$ at equal altitude $h$ over the earth’s surface. Let $R$ be the radius of the earth, and let $\gamma$ be the angle between the positions of $S_1$ and $S_2$ and the center of the earth at a time instant $t$. Then the length of a straight link between $S_1$ and $S_2$ at time $t$ is

$$d_t = 2(R + h)\sin \frac{\gamma}{2} = \sqrt{2(R + h)} \sqrt{1 - \cos \gamma}.$$

We now present a formula for $\cos \gamma$. Since we are only interested in the angle between two satellites, we can simplify the calculations by assuming that the radius of the orbits is equal to one. The proof of the following lemma is also elementary (see Figure 3).

**Lemma 2** Let $S$ be a satellite on an orbit with unit radius and inclination $\alpha_0$. Let $\phi$ be the instantaneous phase of $S$ at a given time instant.

1. If the longitude of the descending node of $S$ is 0°, then its coordinates in the fixed coordinate system are $(\cos \phi, \cos \alpha_0 \sin \phi, -\sin \alpha_0 \sin \phi)$.
2. If the longitude of the descending node of $S$ is $\beta_0$, then the coordinates are $(\cos \beta_0 \cos \phi - \sin \beta_0 \cos \alpha_0 \sin \phi, \sin \beta_0 \cos \phi + \cos \beta_0 \cos \alpha_0 \sin \phi, -\sin \alpha_0 \sin \phi)$.

Note that the satellite reaches the maximum latitude $\alpha_0$ at $\phi = -90°$.

To discuss the length of intersatellite links, we define the phase of a link connecting two satellites $S_1$ and $S_2$ with phases $\phi_1$ and $\phi_2$ to be $\phi = (\phi_1 + \phi_2)/2$. Note that the two satellites do not have to be on the same orbit. The offset of the
connected satellites will be defined as $\Delta \phi = \phi_2 - \phi_1$. The following lemma gives a general formula for the cosine of the angle between two satellites $S_1$ and $S_2$. We omit the proof in this extended abstract.

**Lemma 3** Let $S_1$ and $S_2$ be two satellites with phase offset $\Delta \phi$ that are in two orbits with unit radius and equal inclination $\alpha_0$. Let $\beta_0$ be the angle between the descending nodes of $S_1$ and $S_2$. Assume that the phase of a link connecting satellites $S_1$ and $S_2$ at time $t$ is $\phi_t$. Then the cosine of the angle $\gamma$ between the satellites $S_1$ and $S_2$ at time $t$ is $\cos \gamma = \frac{1}{2} \sin^2 \alpha_0 (1 - \cos \beta_0) (\cos \Delta \phi - \cos 2 \phi_t) + \cos \beta_0 \cos \Delta \phi - \cos \alpha_0 \sin \beta_0 \sin \Delta \phi$.

**Proof Sketch.** The value $\cos \gamma$ is equal to the inner product of the vectors $v_1$ and $v_2$ from the center of the earth to the locations of $S_1$ and $S_2$ at time $t$. The phases of the satellites $S_1$ and $S_2$ are $\phi_1 = \phi_t - \Delta \phi/2$ and $\phi_2 = \phi_t + \Delta \phi/2$. Without loss of generality, assume that the longitude of the descending node of $S_1$ is $0^\circ$, and the longitude of the descending node of $S_2$ is $\beta_0$. Then, applying Lemma 2, $v_1 = (\cos (\phi_1 + \phi_t), \cos \alpha_0 \sin (\phi_1 + \phi_t), -\sin \alpha_0 \sin (\phi_1 + \phi_t))$ and $v_2 = (\cos \beta_0 \cos (\phi_2 + \phi_t) - \sin \beta_0 \cos \alpha_0 \sin (\phi_2 + \phi_t), \sin \beta_0 \cos (\phi_2 + \phi_t) + \cos \beta_0 \cos \alpha_0 \sin (\phi_2 + \phi_t), -\sin \alpha_0 \sin (\phi_2 + \phi_t))$. Taking the inner product of the two vectors $v_1$ and $v_2$ and using some trigonometric identities to simplify the expression gives the result. \hfill \square

We now study the impact of several parameters on the length of intersatellite links. A more complete study, including the impact of the values of $n_h$ and $n_v$, appears in [6]. Consider an interorbital link that connects two satellites with phase offset $\Delta \phi$. Figure 4 plots the length variation of two sample interorbital links as a function of instantaneous phase $\phi_t$. Both links connect two satellites on orbits with inclination $\alpha_0 = 65^\circ$ that are $\beta_0 = 36^\circ$ apart, and whose altitude is 1400 km. The phase offsets of the two links are $\Delta \phi = -18^\circ$ and $\Delta \phi = +18^\circ$. The plots show a significant variation of the link length over time. According to Lemma 1, the link achieves its minimum length when $\cos 2 \phi_t$ is minimized, i.e. when $2 \phi_t = \pm 180^\circ$, or $\phi_t = \pm 90^\circ$. This occurs when the link is at its minimum and maximum latitudes. Similarly, the link length is maximized when the link crosses the equator.

The next parameter upon which we focus is the phase offset between the interconnected satellites $\Delta \phi$. Figure 5 plots the length of an intersatellite link at its minimum and maximum positions as a function of $\Delta \phi$. These positions correspond to $\phi_0 = 90^\circ$ and $\phi_0 = 0^\circ$, respectively. The parameters in Figure 5 are $\alpha_0 = 65^\circ$, $\beta_0 = 36^\circ$, and altitude 1400 km. Note that phase offsets $\Delta \phi < -78^\circ$ and $\Delta \phi > 47^\circ$ are unrealistic since for these values the beam would intersect the earth when the link crosses the equator. The figure shows a fairly strong impact of the phase offset on the link length. The following theorem gives the phase offset that minimizes the length of the links. We omit the proof in this extended abstract.
Theorem 1 The length of an intersatellite link between two satellites on orbits with inclination $\alpha_0$ and with the angle between their descending nodes equal to $\beta_0$ is at any time instant minimized for the phase offset
$$\Delta \phi_{opt} = -2 \arccos \left( \cos \frac{\beta_0}{2} / \sqrt{1 - \sin^2 \alpha_0 \sin^2 \frac{\beta_0}{2}} \right).$$

Proof Sketch. By Lemma 1, minimizing link length is equivalent to maximizing $\cos \gamma_t$. By Lemma 3, for any time instant $t$, $\cos \gamma_t$ is maximized for the phase offset $\Delta \phi$ that maximizes the expression
$$\frac{1}{2} \sin^2 \alpha_0 (1 - \cos \beta_0) \cos \Delta \phi + \cos \beta_0 \cos \Delta \phi - \cos \alpha_0 \sin \beta_0 \sin \Delta \phi. \quad (1)$$
Since this expression does not depend on time $t$, it is sufficient to find the value $\Delta \phi_{opt}$ that minimizes the link length at any given time instant. Consider the time instant when $\phi = -90^\circ$ and the link length achieves its minimum value of 0. This occurs when the connected satellites cross the intersection of their orbits at the same time. The instantaneous phase of the satellite moving north at this moment is equal to $\phi = -90^\circ - \Delta \phi_{opt}/2$. Assuming that the longitude of its descending node is $90^\circ$, all we need to do is to guarantee that the longitude at phase $\phi$ is $-90^\circ + \beta_0/2$. The remainder of the proof involves some manipulation and simplification of trigonometric expressions.

Now we discuss the selection of inter-orbital links for a particular satellite constellation. Let $n_h$ denote the number of orbits and $n_v$ the number of satellites per orbit. For the sake of simplicity, we will assume that all orbits have the same phasing. Since the orbits are assumed to be regularly spaced, $\beta_0 = 360^\circ / n_h$. We have also assumed that the satellites are regularly spaced on their orbits, so we usually cannot achieve the optimum phase difference $\Delta \phi_{opt}$ for satellites on all orbits simultaneously by the following reasoning. If $\Delta \phi_{intra} = 360^\circ / n_v$ is the phase difference between two successive satellites on one orbit, then the phase difference of inter-orbital links $\Delta \phi_{inter}$ must be a multiple of $\Delta \phi_{intra}$, i.e., $\Delta \phi_{inter} = k_0 \times 360^\circ / n_v$ for some integer $k_0$ which we call skew per orbit. To achieve $\Delta \phi_{inter} = \Delta \phi_{opt}$, we need $\Delta \phi_{opt} = k_0 \times 360^\circ / n_v$ or $k_0 / n_v = \Delta \phi_{opt} / 360^\circ$. If both $n_v$ and $k_0$ can be freely chosen, then we can approximate the value $\Delta \phi_{opt} / 360^\circ$ with the ratio $k_0 / n_v$ and the accuracy of the approximation increases with increasing $n_v$. However, increasing $n_v$ means more satellites and an increase in the cost of the system. In practice, the value of $n_v$ is likely to be determined by the desired coverage or by economics and the best we can do towards minimizing the inter-orbital link length is to choose the skew per orbit to be
$$k_0 \approx n_v \frac{\Delta \phi_{opt}}{360^\circ} = \frac{\Delta \phi_{opt}}{\Delta \phi_{intra}}. \quad (1)$$
One can see from Theorem 1 that, in general, the optimum link length requires a non-zero phase offset. Unless $n_v$ is small relative to $n_h$, this results in a negative skew per orbit $k_0$. The resulting topology of the network is then a skewed torus with total skew $k = |k_0| \times n_h$.

Finally, we mention the impact of the inclination on the inter-orbital link.
lengths. Figure 6 shows the link length at its minimum position as a function of the phase offset $\Delta \phi$ for two values of $\alpha_0$. From the figure, we see that the actual values of the link lengths are different, but the shapes of the curves are similar.

4 End-to-end Path Lengths

In this section, we study the distances between arbitrary pairs of satellites. The inter-orbital link lengths vary according to the complex formula given Lemma 3, and we were not able to obtain analytic solutions for path lengths. Instead, we obtained numerical results using the Floyd-Warshall algorithm for all-pairs shortest paths [3]. We first investigate the path lengths in constellations with inclination $\alpha_0 = 65^\circ$ and equal phasing of orbits at the time instant when one of the links achieves its minimum length, i.e. when it is at its maximum latitude. We calculated the maximum and average path lengths between satellites for constellations with $n_h$ and $n_v$ ranging from 5 to 30.

Figure 7 shows the maximum and average path lengths as a function of $k_0$ for a constellation with $n_h = 10$ orbits, $n_v = 30$ satellites per orbit, and altitude 1400 km. We note that only the values of $k_0$ between $-6$ and $+3$ avoid the beams intersecting the earth. The figure shows that a proper choice of the inter-orbital connections can have a significant impact on the maximum and average path lengths.

Next we fix the parameters $n_h = 10$, $n_v = 30$, and choose $k_0$ to minimize the maximum or average path length. Then we vary the inclination and relative phasing of the orbits. The impact of inclination $\alpha_0$ is shown in Figure 8. The path lengths in Figure 8 are calculated at the time instant when one of the links achieves its minimum length. The figure shows that path lengths increase slightly with increasing $\alpha_0$; the variation from $25^\circ$ to $85^\circ$ is about $12.5\%$. However, in practice the inclination is more likely to be determined by the desired coverage than by minimum path lengths. Figure 9 shows the impact of relative phasing of the orbits with $\alpha_0 = 65^\circ$. The $x$-axis shows the relative shift of the phase between two neighbouring orbits as a portion of the angle between two consecutive satellites on one orbit $\Delta \phi_{\text{intra}}$; the value 0 corresponds to equal phasing of orbits. The figure suggests that the path lengths decrease with increasing phase shift and are minimized when the shift is $\Delta \phi_{\text{intra}}/2$.

We end this section with an examination of the relationship between minimum link lengths and minimum path lengths. We start with inclination $\alpha_0 = 65^\circ$ and equally phased orbits. The skew that minimizes link length is given by (1). The skews minimizing maximum and average path lengths were obtained from our numerical results. Figure 10 compares the maximum path length when $k_0$ from (1) is used with its minimum possible value. In the vast majority of cases, choosing the minimum link length also minimizes the maximum path length and the difference in the remaining cases is less than $7\%$ of its optimal value. The results are similar for the average path lengths, as shown in Figure 11. The rela-
tive error is even smaller, being under 1.5%. The results for other inclinations are summarized in Figure 12. We calculated the path lengths for inclinations between 25° and 85° for values of \( n_h \) and \( n_v \) which are multiples of 5 between 5 and 30. Figure 12 shows the percentage of cases when the skew chosen according to formula (1) does not minimize the maximum and average path lengths. Overall, the results suggest that minimizing inter-orbital link lengths is a good heuristic for minimizing end-to-end path lengths, especially for larger inclinations.

5 Modelling Intersatellite Links

In many terrestrial networks, the differences among the propagation delays associated with the links are small enough to be ignored, so it is reasonable to use models that assume all links have the same length. We call such a model a \( 1 \)-uniform model. In a LEO network, two satellites on the same orbit keep a constant relative position and distance, so the length of intra-orbital links does not vary significantly with time and position of the satellites. If all orbits contain the same number of regularly spaced satellites, then the length of all intra-orbital links is the same, so we model them by just one value. In contrast, the inter-orbital links have constantly changing lengths according to the complex formulae from Lemmas 1 and 3. Because of the complexity of the expressions, we propose two ways to approximate them. The first method is a linear approximation of the curve between its minimum and maximum. The second method uses a constant value equal to the midpoint between the minimum and maximum.

From Lemma 3 we see that an inter-orbital link achieves its minimum when \( \cos 2\phi_t = -1 \), and its maximum when \( \cos 2\phi_t = +1 \). Therefore, the minimum length of the link is \( d_{\text{min}} = \sqrt{2(R + h)\sqrt{1 - \cos \gamma_{\text{min}}}} \), where \( \cos \gamma_{\text{min}} = \frac{1}{2} \sin^2 \alpha_0(1 - \cos \beta_0)(1 + \cos \Delta \phi) + \cos \beta_0 \cos \Delta \phi - \cos \alpha_0 \sin \beta_0 \sin \Delta \phi \). Similarly, the maximum length is \( d_{\text{max}} = \sqrt{2(R + h)\sqrt{1 - \cos \gamma_{\text{max}}}} \), where \( \cos \gamma_{\text{max}} = \frac{1}{2} \sin^2 \alpha_0(1 - \cos \beta_0)(-1 + \cos \Delta \phi) + \cos \beta_0 \cos \Delta \phi - \cos \alpha_0 \sin \beta_0 \sin \Delta \phi \). The linear approximation uses a linear segment between each consecutive minimum and maximum as shown in Figure 13. The corresponding formulae can be written as

\[
\begin{align*}
\hat{d}_i^{(l)} &= \begin{cases} 
  d_{\text{max}} - (d_{\text{max}} - d_{\text{min}}) \frac{\phi_t}{90} & \text{for } \phi_t \in [-180°, -90°] \\
  d_{\text{min}} + (d_{\text{max}} - d_{\text{min}}) \frac{\phi_t}{90} & \text{for } \phi_t \in [-90°, 0°] \\
  d_{\text{max}} - (d_{\text{max}} - d_{\text{min}}) \frac{\phi_t}{90} & \text{for } \phi_t \in [0°, 90°] \\
  d_{\text{min}} + (d_{\text{max}} - d_{\text{min}}) \frac{\phi_t}{90} & \text{for } \phi_t \in [90°, 180°].
\end{cases}
\end{align*}
\]

The constant approximation uses a constant value \( \hat{d}_i^{(c)} = \frac{1}{2}(d_{\text{min}} + d_{\text{max}}) \) for all time instants \( t \) as shown in Figure 13. This method disregards the time-varying aspect of the network, but it does capture the difference between the two types of links—inter-orbital and intra-orbital. Moreover, as we will see below, it sometimes outperforms the linear approximation. Replacing the length of inter-orbital links with one constant value gives a model that we call \( 2 \)-uniform.
The rest of this section discusses the error introduced by the approximations. Figures 14 and 15 show the absolute values of the relative error of the average path length for $\alpha = 65^\circ$, equally phased orbits, and the time instant when one of the links achieves its minimum length. The ranges of $n_h$ and $n_v$ are 5 to 30 and the skew per orbit $k_0$ was chosen according to formula (1). From the figures we see that the error for both the linear and constant approximation is less than 15%. It is interesting to note that the “ridge” of the largest error for the linear approximation roughly corresponds to the cases when the inter-orbital skew is close to the optimum value $\Delta \varphi_{opt}$ while the error for the constant approximation achieves its lowest values in this region. The results for the maximum path length (the diameter of the network), are qualitatively similar.

Next, we illustrate the impact of the inclination on the accuracy of the approximations. We calculated the error of the approximations for values of $n_h$ and $n_v$ that are multiples of 5 between 5 and 30 for various values of inclination. Then we obtained both the maximum error and mean error by averaging over all values of $n_h$ and $n_v$. Figure 16 show the maximum and average of the absolute value of the relative error of the average path length for both the linear and constant approximation. A similar plot for the maximum path length is shown in Figure 17. The plots show that, on average, the linear approximation outperforms the constant one. They also show that the error of the constant approximation increases sharply for higher inclinations.

6 Conclusions

In this paper, we have proposed simple models for the study of inclined LEO satellite networks. Our models are based on analytical and empirical investigations of link lengths and path lengths. A comparison of our models with the analytical and empirical results shows that our models introduce less than 15% error and for many choices of satellite constellation, the error is much smaller. We have used the constant approximation model in [7] to study the gossiping communication pattern. Our study has concentrated on the choice of intersatellite links and on link lengths and path lengths. There are many other parameters that we have not yet analyzed.

References


   http://www.ee.surrey.ac.uk/Personal/L.Wood/constellations/tables/.


Peter Gvozdjak recently completed his Ph.D. in the School of Computing Science, Simon Fraser University. He is currently with Motorola Canada Ltd., 11411 Number Five Road, Richmond, B.C., Canada, V7A 4Z3. E-mail: peterg@cs.sfu.ca

Joseph Peters is a professor in the School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada, V5A 1S6. E-mail: peters@cs.sfu.ca
Figures

Unless noted otherwise in the captions, the parameter values in the following figures are $\alpha_0 = 65^\circ$, $\beta_0 = 36^\circ$, $n_h = 10$, $n_v = 30$, the altitude is 1400 km, and the skew per orbit $k_0$ is chosen using formula 1 to minimize the intersatellite link lengths.

Figure 1: A 3-skewed torus of size $4 \times 6$

Figure 2: An interconnection for an inclined satellite constellation

Figure 3: General position of a satellite

Figure 4: Length variation of inter-orbital links as a function of instantaneous phase $\phi_t$ for phase offsets $-18^\circ$ and $+18^\circ$

Figure 5: Impact of $\Delta\phi$ on the length of intersatellite links at their minimum and maximum positions
Figure 6: Impact of inclination $\alpha_0$ on the length of intersatellite links at minimum position. Values of $\alpha_0$ are 65° and 55°. Values of $\beta_0$ are 90° and 36°.

Figure 7: Impact of skew per orbit $k_0$ on maximum and average path length.

Figure 8: Impact of inclination $\alpha_0$ on average and maximum path lengths.

Figure 9: Impact of relative phasing between orbits on average and maximum path lengths.

Figure 10: Relative difference between maximum path length for the skew optimizing intersatellite link length and its minimum value.

Figure 11: Relative difference between average path length for the skew optimizing intersatellite link length and its optimal value.
Figure 12: Percentage of cases when the skew minimizing link length is different from the skew minimizing the maximum and average path for \( n_h \) and \( n_v \), multiples of 5 in the range 5 to 30

Figure 13: Linear and constant approximations of inter-orbital link lengths with \( \Delta \varphi = -18^\circ \)

Figure 14: Absolute value of the relative error of average path length for the linear approximation

Figure 15: Absolute value of the relative error of average path length for the constant approximation

Figure 16: Relative error of average path length for the linear and constant approximations as a function of inclination

Figure 17: Relative error of maximum path length for the linear and constant approximations as a function of inclination
On the Graphs of McKay-Miller-Širáň

PAUL R. HAFNER

University of Auckland, New Zealand

Abstract

McKay, Miller and Širáň introduced a family of graphs of order $2q^2$, where $q$ is a prime power. As graphs of diameter 2, they are of considerable interest in the context of the degree-diameter problem, especially since they are vertex transitive if $q \equiv 1 \mod 4$.

We investigate their structure in the case that $q \equiv 1 \mod 4$ and determine their automorphisms.

Keywords

McKay-Miller-Širáň graphs, degree-diameter problem, graph automorphisms

1 Introduction

McKay, Miller and Širáň [8] used an ingenious voltage graph construction to define an infinite family of graphs $H_q$ of order $2q^2$, where $q$ is a prime power. These graphs have diameter 2. If $q \equiv 1 \mod 4$ they are non-Cayley vertex transitive graphs of degree $(3q-1)/2$. The graph $H_5$ is the Hoffman-Singleton graph of degree 7, order 50, and girth 5. The graph $H_9$ has degree 13 and order 162, which is tantalisingly close to the Moore bound of 170. The construction provides an infinite family of lower bounds in the degree-diameter problem, as well as some graphs of largest known order for diameter 2 and given degree [4, 8].

A simplified construction of the graphs $H_q$, also using voltages, has recently been given by Šiagiova [10] for the case $q \equiv 1 \mod 4$.

We will elucidate the structure of $H_q$ for $q \equiv 1 \mod 4$ and determine the automorphisms of $H_q$ for $q > 5$.

2 The Graphs $H_q$

Let $p$ be a prime, $n$ a natural number and assume that $q = p^n \equiv 1 \mod 4$. We denote the field with $q$ elements by $F_q$ and let $\xi$ be a primitive element of $F_q$. The multiplicative group of $F_q$ will be denoted by $F_q^*$.

* This work has been supported in part by a University of Auckland Research Grant.
The Paley graph \( QR(q) \) of order \( q \) has the elements of \( F_q \) as vertices, and \( u \in F_q \) is adjacent to all \( u + x^2, x \in F_q^* \). It is a Cayley graph of valency \((q - 1)/2\). Moreover, \( QR(q) \) is self-complementary, strongly regular, and of diameter 2 (a complementing permutation is given by \( x \mapsto \xi x \)). For \( q = 5 \) we get the 5-cycle, but for \( q > 5 \) every edge of \( QR(q) \) belongs to a triangle [5]. The automorphism group of the Paley graphs was determined by Muzychuk [9]: \( \text{Aut}(QR(q)) \) consists of all mappings \( x \mapsto m^2 x^{d'} + b, m \in F_q^*, \ell \in \{ 1, \ldots, n \}, b \in F_q \), and is thus a subgroup of index 2 of \( \text{AGL}(1, q) \). The group \( \text{AGL}(1, q) \) (where \( x \mapsto mx^{d'} + b, m \in F_q^* \)) also acts on \( QR(q) \); its transformations are either automorphisms or complementing permutations of \( QR(q) \).

**Definition 1** Let \( p \) be a prime, \( n \) a positive integer, such that \( q = p^n \equiv 1 \mod 4 \). The graphs \( H_q \) of McKay, Miller and Sírañ [8, 10] have the vertex set \( \mathbb{Z}_2 \times F_q \times F_q \), and adjacency is defined by

\[
(i, a, b) \quad \text{is adjacent to} \quad (i, a, b + \xi x^2), \quad x \in F_q^*, \quad (1)
\]

and to

\[
(i + 1, a + x, b + (-1)^j x^2), \quad x \in F_q. \quad (2)
\]

The following observations expand on the description given in [8].

**Remark:** Since \( -1 \) is a square in \( F_q \), the adjacency relation defined above is symmetric, and the degree of the resulting graph is \( q + (q - 1)/2 = (3q - 1)/2 \).

**Remark:** The adjacencies (1) in Definition 1 show that for fixed \( a \), the \( q \) vertices \((0, a, b), b \in F_q \), form a Paley subgraph \( P_a \) of \( H_q \). Similarly, for fixed \( a \), the \( q \) vertices \((1, a, b), b \in F_q \) form a subgraph \( Q_a \) isomorphic to a Paley graph (use the fact that \( QR(q) \) is self-complementary).

**Definition 2** We will denote the subgraph of \( H_q \) induced by all vertices of Paley subgraphs \( P_a \), \( a \in F_q \), by \( V_0 \). Similarly, \( V_1 \) is the subgraph of \( H_q \) induced by all vertices of \( Q_a \), \( a \in F_q \). We also use the notation \( B_0 \) for the set \( \{ P_a: a \in F_q \} \).

**Remark:** For fixed \( a \) and \( x = 0 \) we see from the adjacencies (2) in Definition 1 that the \( 2q \) vertices \((0, a, b) \) and \((1, a, b) \) (i.e. the vertices of \( P_a \cup Q_a \)) form a subgraph of diameter 2 (because \((0, a, b) \) is adjacent to \((0, a, b') \) if and only if \((1, a, b) \) is not adjacent to \((1, a, b') \)).

Since the mappings \( a \mapsto a \pm x^2 \) are automorphisms of \( QR(q) \) as well as of its complement, the preceding reasoning also shows that the vertices of \( P_r \cup Q_s \) define a subgraph of diameter 2 and order \( 2q \) for all \( r, s \in F_q \).

**Remark:** There are no edges between vertices of \( P_a \) and vertices of \( P_d \) for \( a \neq d \). That \( H_q \) has diameter 2 now follows from the observation that any two vertices \((0, a, b) \) and \((0, a', b') \) with \( a \neq a' \) have a unique common neighbour (in some \( Q_x \)). This is the key element in the construction in [8]. See section 3.

We might visualise the graph \( H_q \) as in Figure 1, which tries to indicate that the only edges are within the Paley subgraphs, or between Paley subgraphs on different sides.
Even though the Hoffman-Singleton graph is the odd one out in the family of graphs $H_q$, the construction of $H_q$ is now seen to be a beautiful generalisation of the familiar ‘pentagons and pentagrams’ construction (attributed to N. Robertson in [3]) for the Hoffman-Singleton graph.

**Remark:** If $q > 5$, the only edges of $H_q$ which are not contained in a triangle are between vertices of $V_0$ and vertices of $V_1$. Given that $H_q$ is vertex transitive [8], this implies that $\{V_0, V_1\}$ is a block system for the automorphism group. The connected components of $V_0$ and of $V_1$ are the Paley subgraphs $P_a, Q_b$, and therefore $\{P_a, Q_b : a, b \in F_q\}$ is another block system for the automorphism group of $H_q$.

**Remark:** Finally we note that adjacency in $H_q$ is defined essentially by taking differences in components 2 and 3 of $(i, a, b)$. This indicates some obvious automorphisms of $H_q$, namely $(i, a, b) \mapsto (i, a + s, b + t), s, t \in F_q$.

### 3 Midpoints

As mentioned before, the following simple lemma is the essential part of the verification that $H_q$ has diameter 2.

**Lemma 1** If $u, v$ are vertices in $V_0$ belonging to distinct Paley subgraphs, then there exists a unique vertex $M(u, v) = (1, a, b) \in V_1$ such that $m$ is adjacent to both $u$ and $v$.

**Proof.** A walk from $(0, a, b)$ via $V_1$ back to $(0, c, d)$ looks like

$$(0, a, b) \rightarrow (1, a + z_1, b + z_2^3) \rightarrow (0, a + z_1 + z_2, b + z_1^2 - z_2^3) = (0, c, d).$$
This leads to the system of linear equations
\begin{align*}
z_1 + z_2 &= c - a \\
z_1 - z_2 &= \frac{d - b}{c - a}
\end{align*}
(3)

It is not necessary to solve this system to obtain the desired result.

We will refer to \( M(u, v) \) as the midpoint of \( u \) and \( v \).

If one solves the system of equations (3), one finds that the midpoint of \( u = (0, a, b) \) and \( v = (0, c, d) \) is

\[
\left(1, \frac{c^2 - a^2 + d - b}{2(c - a)}, \frac{(c - a)^2 + b + d}{2(c - a)} - \frac{bd}{(c - a)^2}\right).
\]

Of course, the definition of midpoint has its obvious counterpart for vertices in \( V_1 \) (with midpoints in \( V_0 \)). All of the following properties of midpoints have their natural analogs when the roles of \( V_0 \) and \( V_1 \) are reversed.

**Remark:** If \( u \in P_a \) is fixed and \( b \neq a \) then

\[
\{M(u, v) : v \in P_b\}
\]
contains precisely one element of each \( Q_j, a \in F_q \). In other words: given a fixed vertex \( u \in P_a \) and a Paley subgraph \( P_b \) with \( a \neq b \), we obtain a bijection between vertices of \( P_b \) and Paley subgraphs \( Q_j \).

Another way of viewing this is:

**Lemma 2** Given two distinct Paley subgraphs \( P_a, P_b, P_c \), two vertices \( u, z \in P_b \), and a vertex \( x \in P_a \), there exists a unique \( v \in P_a \) such that the midpoint \( M(z, v) \) is contained in the same Paley subgraph \( Q_j \) as \( M(u, x) \).

**Remark:**

The bijection established above can be used to define the structure of a Paley graph on the Paley subgraphs \( Q_j \) as vertices (but adjacency is not defined arithmetically in the subscripts of \( Q \), rather by adjacency in \( P_b \)). This structure is almost independent of the choice of the vertex \( u \): the structure does not change if \( u \in P_a \) is replaced by \( u' \) in the same Paley subgraph \( P_a \). If \( u \) is replaced by \( u' \) in \( P_a \) the Paley structure either remains unchanged or switches to its complement.

The following result will also be used.

**Lemma 3** Let \( P_a, P_b, P_c \) be three distinct Paley subgraphs, \( u \in P_a \), \( x \in P_b \), \( z \in P_c \). If \( M(z, x) = M(z, u) \) then \( M(z, x) = M(u, x) \).
4 Automorphisms of $H_q$

Using software like MAGMA [2, 7] one can determine the automorphism group of $H_q$ for small $q$. The following table shows some of the orders.

<table>
<thead>
<tr>
<th>$q$</th>
<th>$\text{Aut}(H_q)$</th>
<th>order of $H_q$</th>
<th>valency</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>252000</td>
<td>50</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>93312</td>
<td>162</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>316368</td>
<td>338</td>
<td>19</td>
</tr>
<tr>
<td>17</td>
<td>1257728</td>
<td>578</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>18000000</td>
<td>1250</td>
<td>37</td>
</tr>
</tbody>
</table>

The large automorphism group of $H_5$ is explained in part by the fact that the Hoffman-Singleton graph contains 126 sets of 10 disjoint 5-cycles (= Paley subgraphs $QR(5)$) [1]. By contrast, we have seen that if $q > 5$, there is a unique set of 2q disjoint Paley subgraphs $QR(q)$ in $H_q$.

Throughout this section it will be assumed that $q > 5$, excluding the Hoffman-Singleton graph from consideration.

**Definition 3** We denote the automorphism group of $H_q$ by $\text{Aut}(H_q)$. The subgroup consisting of the automorphisms which preserve the block $V_0$ is denoted by $\text{Aut}_0(H_q)$.

Since $\text{Aut}(H_q)$ is vertex transitive [8], $\text{Aut}^+(H_q)$ is a subgroup of index 2 in $\text{Aut}(H_q)$. We will concentrate on $\text{Aut}^+(H_q)$.

Some examples of automorphisms are listed in the following lemma.

**Lemma 4** For $m \in F_q^*$, $s,t \in F_q$, $\ell \in \{1,\ldots,n\}$ the mappings

$$ (i,a,b) \mapsto \left( i, ma^{\ell} + s, m^2 b^{\ell} + t \right) $$

are automorphisms in $\text{Aut}^+(H_q)$ ($n$ denotes the degree of $F_q$ over its prime field).

The proof is a straightforward verification. The mappings (4) share the property that the pairs $(P_a, Q_a)$ are moved to pairs $(P_{a'}, Q_{a'})$, i.e. the Paley subgraphs in $V_0, V_1$ move in tandem.

The automorphisms (4) form a subgroup of $\text{Aut}^+(H_q)$. We are interested in the smaller subgroup $K$ of the mappings

$$ (i,a,b) \mapsto (i, ma + s, m^2 b) $$

which is isomorphic to $AGL(1,q)$, and operates 2-transitively on the block system $B_0$. It follows from this transitivity that if two distinct Paley subgraphs $P_a$ and $P_b$ are given, we can write every automorphism $\phi \in \text{Aut}^+(H_q)$ as a product $\phi = \sigma \tau$, with
where $\sigma \in K$, and $\tau$ belongs to the subgroup $L_{a,b} < \text{Aut}^{+}(H_q)$ consisting of those automorphisms which fix both $P_a$ and $P_b$.

These latter automorphisms are completely determined by their restriction to one of the distinguished subgraphs, $P_a$, say, and the image of one element $u \in P_b$, as expressed in the next lemma.

**Lemma 5** Let $\sigma : P_a \to P_a$ be an automorphism of a Paley subgraph of $H_q$, $u, u' \in P_b$, $u \neq b$. Then there exists a unique automorphism $\tilde{\sigma}$ in $\text{Aut}^{+}(H_q)$ such that $\tilde{\sigma}|_{P_a} = \sigma$ and $\tilde{\sigma}(u) = u'$.

**Proof.** The following construction shows that there is at most one extension of $\sigma$ to $\tilde{\sigma}$. Assume first that $z \in P_b$.

Choose $x \in P_a$. We can find a unique $v \in P_a$ such that $M(z, v)$ and $M(u, x)$ lie in the same Paley subgraph $Q_j$ (cf. Lemma 2).

Now consider $\sigma(v) \in P_a$ and find the unique $z' \in P_b$ such that $M(z', \sigma(u)) = M(u', \sigma(x))$. Since automorphisms must respect midpoints, the only way to define $\tilde{\sigma}$ is by putting $\tilde{\sigma}(z) = z'$.

A similar process, making use of the extension of $\sigma$ to $P_b$, defines $\tilde{\sigma}$ on all of $V_0$. To define $\tilde{\sigma}$ on $V_1$ we recall that every element $z \in V_1$ is a midpoint $M(x, y)$ for some $x \in P_a, y \in P_b$. We must define $\tilde{\sigma}(z) = M(\sigma(x), \tilde{\sigma}(y))$. Lemma 3 is useful here.

After verifying that the images as constructed are well-defined, and that adjacencies are preserved, the proof is complete.

We summarise some of our results in the following theorem.

**Theorem 1** Let $p$ be a prime, $n$ a positive integer, $q = p^n \equiv 1 \mod 4$ and assume that $q > 5$. Denote by $\text{Aut}^{+}(H_q)$ the group of those automorphisms of the McKay-Miller-Širán graph $H_q$ which fix the block system $(V_0, V_1)$, by $L_{0,1}$ the subgroup of those automorphisms which stabilise both $P_0$ and $P_1$, and let $K$ be the subgroup of the mappings (5). Then

(a) every automorphism in $\text{Aut}^{+}(H_q)$ can be expressed uniquely as a product $\sigma \tau$ where $\sigma \in K$ and $\tau \in L_{0,1}$.

(b) $|\text{Aut}^{+}(H_q)| = nq^3(q - 1)^2/2$ (and hence $|\text{Aut}(H_q)| = nq^3(q - 1)^2$).

**Proof.** (a) The existence of such a decomposition was pointed out above, and the uniqueness results from the fact that $K \cap L_{0,1} = \text{id}$, since the only affine transformation in $AGL(1, q)$ which fixes two points is the identity.

(b) $K$ has $(q - 1)q$ elements, and it follows from Lemma 5 that $L_{0,1}$ has $n(q - 1)q^2/2$ elements ($n(q - 1)q/2$ possibilities for the action on $P_0$ and $q$ possibilities to assign an image to a given element of $P_1$).
In conclusion we note that the groups we encountered are of a geometric nature. This will be discussed elsewhere [6].

Acknowledgement. The author is grateful to Eamonn O’Brien for helpful discussions.

References


Paul Hafner is with the Department of Mathematics, University of Auckland, Auckland, New Zealand. E-mail: hafner@math.auckland.ac.nz
QoS-Competitive Video Buffering

ALEXANDER KESSELMAN
Tel-Aviv University

YISHAY MANSOUR
Tel-Aviv University

Abstract

Many multimedia applications require transmission of streaming video from a server to a client across an internetwork. In many cases loss may be unavoidable due to congestion or heterogeneous nature of the network. We explore how discard policies can be used in order to maximize the quality of service (QoS) perceived by the client. In our model the QoS of a video stream is measured in terms of a cost function, which takes into account the discarded frames. In this paper we consider online policies for selective frame discard and analyze their performance by means of competitive analysis. In competitive analysis the performance of a given online policy is compared with that of an optimal offline policy. In this work we present competitive policies for a wide range of cost functions, describing the QoS of a video stream.

Keywords
competitive analysis, selective frame discard, multimedia, QoS metrics

1 Introduction

The emergence of high-speed networks facilitates many multimedia applications that rely on the efficient transfer of compressed video. However, in reality both network bandwidth and buffering capacity are likely to be limited. Under such circumstances there may be a situation in which a feasible transmission schedule is impossible, i.e., any transmission necessarily incurs loss of data. In this case instead of being denied service, clients may choose lower quality video streams with occasional frame loss. This is a natural situation in today’s Internet where the network may not have sufficient bandwidth to support the peak rate of a video stream [6], or available bandwidth may fall below the requirement at the middle of a video transmission.

When delivering a video stream across a resource-constrained network, a naive approach is to discard frames with no awareness of the video stream properties. As a result the QoS perceived by the user may degrade drastically, even for
small amounts of loss (e.g. tail drop of consecutive frames). In this paper we study intelligent selective frame discard policies, which take into account application-specific properties.

**Previous work.** Recently a number of packet discarding schemes incorporating application-specific information have been proposed. In [9] appears a simple strategy called Frame-Induced Packet Discarding, in which upon detection of loss of a threshold number of packets belonging to a video frame, the network attempts to discard all the remaining packets of that frame. In [3], [12] the problem of optimizing the quality of the transmitted video for a given cost function by jointly selecting the source rate and the channel rate has been considered with leaky bucket constraints. Our work differs from theirs in that we are trying to optimize the QoS perceived by the user, rather than minimizing loss in general.

In [18] offline algorithms for optimal selective frame discard have been considered. The notion of selective frame discard at the server has been introduced and the optimal selective frame discard problem using a QoS-based cost function has been defined. There appear various algorithms for selective frame discard for stored video delivery across a resource-constrained network. In contrast with [18], we deal with online problem, when no information regarding the video stream is known a-priori.

In [8], the question of video smoothing is studied, and an upper bound is derived on the competitive ratio of the greedy policy throughput for FIFO model. The result was later improved [4], where they also introduced a new bounded packet delay model. The work of [5] studies the competitive analysis of the loss value rather than the throughput value. Competitive analysis of jitter control online algorithms appears in [7].

**Our work.** We measure the performance of our algorithms using competitive analysis [13], [1]. In competitive analysis the performance of an online policy is compared with those of an optimal offline policy, which knows in advance the entire sequence of frame arrivals. Competitive analysis is a natural approach for Internet traffic, which is unpredictable and chaotic [14]. In this paper we study competitive online policies for intelligent selective frame discard. An intelligent frame discard policy should take in account the application-specific properties in its decisions to discard particular frames, minimizing the QoS degradation perceived by the user. We consider video streams without inter-frame dependencies, such as JPEG video streams. In order to measure the QoS perceived by the user we assume a general cost function and concentrate on a subclass called well-behaved. The cost function should reflect the playback discontinuity at the client, therefore, the cost of a discarded frame should depend either on the distance to the adjacent but non-consecutive discarded frame or on the position of the frame within a sequence of consecutive discarded frames. The QoS level function is the sum of the costs of the discarded frames. We consider various video stream settings, and our main result is that the competitive ratio of the Greedy Policy is bounded by a constant for all of these settings. In addition to the global QoS function we con-
sider local metrics, such as the minimal distance between two discarded frames and maximal length of a sequence of consecutive discarded frames. The Greedy Policy is shown to be competitive with regards to these metrics as well.

The rest of this paper is organized as follows. Section 2 contains the model description. The scheduling policy is defined in Section 3. Section 4 contains analysis of fixed size frames. Our conclusion is in Section 5.

2 Model Description

We consider a system with a single FIFO buffer delivering a video stream. We assume that the buffer can hold exactly $M$ frames when considering fixed size frames and at least $M$ frames of maximal size when considering variable size frames. Frames may arrive to the queue at any time while send events are synchronized with time. We divide time into slots so that during a slot one video frame is sent. We assume that differences between transmission times of video frames due to variations in frame size are negligible.

A queue policy has to decide for each incoming frame whether it should be either accepted or rejected (subject to the buffer capacity constraints). A frame in the buffer may be also preempted by the policy. However, all the transmitted frames must be sent in the order they arrived.

Now we describe the traffic-shaping policy, i.e., the regulation of the rate at which a flow is allowed to inject packets into the QoS network. We consider leaky bucket traffic-shaping mechanism in which only a fixed amount of traffic is admitted to the network. Excess traffic is held in a queue until either it can be accommodated or must be discarded. Wrege et al. [16], [17] investigate providing deterministic performance guarantees to VBR video using multiple leaky buckets traffic constraint function. A $(\sigma, \rho)$ leaky bucket model has a burst parameter $\sigma$ and a rate parameter $\rho$, and assumes that during a time interval of length $t$ there are at most $\rho \cdot t + \sigma$ frame arrivals. Our results vary with the size of $\sigma$.

**Definition 1** An arrival sequence is generated by $(\sigma, \rho)$ source if during any time interval $[t_1, t_2]$ at most $\rho \cdot (t_2 - t_1) + \sigma$ frames arrive.

In our model each discarded frame has the corresponding cost. The cost of a discarded frame depends on the positions of previously discarded frames. The goal is to design a policy that minimizes the total cost of the system. Next we give a formal definition of the notation used. We start with a notation for a sequence of frames.

**Definition 2** A sequence of frames is denoted by $S$. The start and the finish time of $S$ are denoted by $t_{\text{start}}$ and $t_{\text{final}}$.

Next we set the notation to indicate lost frames.
Definition 3 The loss indicator $X_f(t)$ indicates whether frame $f$ was discarded by time $t$, i.e., $X_f(t)$ is 1 if $f$ was discarded prior to time $t$ and 0 otherwise.

At this point we define the terminology that is used to describe dependencies between discarded frames.

Definition 4 The closest discarded frames among frames either preceding or succeeding to a frame $f$ are said to be adjacent discarded frames of $f$. The distance between two adjacent discarded frames is the number of transmitted frames between them. Two discarded frames are said to be consecutive if they are at zero distance. Let a lost block be a maximal sequence of consecutive discarded frames.

Now we define the parameters which determine the cost of a discarded frame.

Definition 5 Let $d(f,t)$ be the distance between a frame $f$ and the closest adjacent discarded frame at time $t$. Let $pos(f,t)$ be the position of a frame $f$ within a sequence of consecutive discarded frames at time $t$, or 0 if $f$ does not participate in such a sequence, i.e., if the previous frame to $f$ is not discarded by time $t$.

Once we have the necessary notation, we can define the structure of the cost function.

Definition 6 The cost of a discarded frame $f$ at time $t$ is denoted by $c_{\phi_1,\phi_2}(f,t)$. If $d(f,t) > 0$ then $c_{\phi_1,\phi_2}(f,t) = \phi_1(d(f,t))$, otherwise $c_{\phi_1,\phi_2}(f,t) = \phi_2(pos(f,t))$.

We define the cost incurred by a policy over a sequence of frames as the sum of the costs of discarded frames.

Definition 7 The cost of a policy $A$ while scheduling $S$ is the sum of the costs of the discarded frames, i.e., $L_A(S) = \sum_f X_f(t^{final})c_{\phi_1,\phi_2}(f,t^{final})$.

Next we define a subclass of well-behaved cost functions. In order to provide measure of playback discontinuity, a cost function should take two aspects into consideration: the length of a sequence of consecutive discarded frames and the spacing between two adjacent, but non-consecutive discarded frames (see [10]). The motivation behind the definition is as follows. For a single discarded frame the QoS degradation is minor while for sequences of consecutive discarded frames it is significant. Therefore, we would like the cost of a single discarded frame to be bounded by constant and the cost of a discarded frame within a sequence to be bounded by a constant factor of its position within a sequence of discarded frames
\footnote{1Notice that the cost of a lost block would depend quadratically on its length.}. The cost of a frame within a sequence must be at least as high as the cost of a single discarded frame since it is always preferable, with respect to the video stream QoS, to drop a frame whose immediate neighbors are not discarded. The
QoS perceived by the user degrades as the distance between two discarded frames decreases or the length of a lost block increases. So we require a cost function to be monotonically increasing with: (1) decreasing distance between two non-consecutive discarded frames (2) increasing position of a discarded frame within a sequence of consecutive discarded frames. The best QoS is achieved when the loss is distributed as evenly as possible over the frames sequence, therefore the cost function should be minimal in this case. We require that the cost function be convex so that its minimum is established when all dropped frames are evenly spaced.

**Definition 8** We say that a cost function \( c_{\phi_1, \phi_2} \) is well-behaved iff there exist positive constants \( L, U, L', U' \), \( 0 < L \leq U \leq L' \leq U' \), satisfying the following constraints with respect to \( \phi_1 \):

1. **bounded cost**: \( \forall d : L \leq \phi_1(d) \leq U \)
2. **anti-monotonicity**: if \( x < y \) then \( \phi_1(x) > \phi_1(y) \)
3. **convexity**: \( \forall x, y : \phi_1((x+y)/2) \leq (\phi_1(x) + \phi_1(y))/2 \)

and with respect to \( \phi_2 \):

1. **bounded linear cost**: \( \forall l : L' \cdot l \leq \phi_2(l) \leq U' \cdot l \)
2. **monotonicity**: if \( x > y \) then \( \phi_2(x) > \phi_2(y) \)
3. **convexity**: \( \forall x, y : \phi_2((x+y)/2) \leq (\phi_2(x) + \phi_2(y))/2 \)

For example, a simple well-behaved cost function is: \( \phi_1(d) = 1 + 1/d \) and \( \phi_2(l) = 4l \), where \( L = 1, U = 2, L' = 3 \) and \( U' = 5 \).

Now we turn to define competitive analysis. In competitive analysis (see [13]) the performance of an online policy is compared with those of the optimal policy \( \text{OPT} \), which knows in advance the entire sequence of frame arrivals. The competitive ratio is the minimum over all input sequences of the ratio between the cost incurred by \( \text{OPT} \) and the the cost incurred by the given online policy. More formally,

**Definition 9** A policy \( A \) is \( c \)-competitive if for every sequence of frames \( S \), \( c \cdot L_A(S) \leq L_{\text{OPT}}(S) \). (Note that \( 0 \leq c \leq 1 \)). In such a case we also say that \( A \) is competitive.

Notice that a competitive online policy must not discard any frame from an input sequence \( S \) if \( \text{OPT} \) does not discard any frame from \( S \).
3 Greedy Policy

We define a natural “Greedy Policy”. The state of the policy depends on the previous decisions of the policy to discard particular frames.

**Greedy Policy**: Each time when the buffer is full and a frame arrives the policy discards a frame that minimizes the sum of the increase in cost of previously discarded frames plus the cost of the discarded frame itself.

3.1 Scheduling Example

Intuitively, when Greedy selects a frame to be discarded, the decision is optimized locally with respect to the current state of the system. On the other hand *OPT* optimizes its decision globally with respect to the entire schedule. We study the performance of the Greedy policy scheduling moderate and large bursts. Let us consider a system consisting of a buffer that is able to hold 3 frames. We define the *schedule* of a policy to be the sequence of arrive and send events (if any) for a stream of video frames.

**Moderate bursts**: For moderate bursts Greedy does not drop consecutive frames, as we prove later. Suppose that at time 0 the buffer is empty and a burst of 3 frames arrives. During the following *t* time units one frame is sent by the Greedy policy and one frame arrives. Finally, at time *t* a burst of 4 frames arrives. In this case Greedy would drop 3 frames among the last six frames because its buffer is full. At the same time *OPT* would evenly distribute the loss over the whole sequence, which can be done since the buffer is not empty throughout this time interval. The resulting video streams appear on Figure 1 (dropped frames are marked by ×).

![Figure 1: Outcoming video streams of Greedy and OPT (moderate bursts).](image)

**Large bursts**: In presence of massive bursts both *OPT* and Greedy are forced
to drop consecutive frames. In this case the loss of $OPT$ is uniformly divided between lost blocks while Greedy may create larger lost blocks. However, they are balanced later by future Greedy discards. Suppose that at time 0 the buffer is empty and a burst of 3 frames arrives. During the following $t$ time units one frame is sent by the Greedy policy and one frame arrives. Finally, at time $t$ a burst of 8 frames arrives. Notice that Greedy would have its buffer full and $OPT$ would have its buffer empty at this time. Now Greedy and $OPT$ would discard 7 and 5 frames, respectively. The resulting video streams are shown on Figure 2.

![Figure 2: Outcoming video streams of Greedy and $OPT$ (large bursts).](image)

### 4 Uniform Size Frames

In this section we consider video streams without inter-frame dependencies, such as JPEG video streams, and assume that the buffer can hold exactly $M$ frames. We use the following definitions regarding the local properties of a schedule.

- Let the maximal distance between two adjacent discarded frames produced by any policy when $k$ frames are lost in time interval of length $s$ be denoted by $d_{OPT}(k,s)$.

- Let the maximal length of a lost block produced by any policy when $k$ frames are lost in time interval of length $s$ be denoted by $l_{OPT}(k,s)$.

#### 4.1 Moderate Burst Size

We consider video streams generated by a $(\sigma, p)$ source with $\sigma \leq \frac{2}{3}M$. We show that the competitive ratio of the Greedy Policy is at least $L/U$. Although there exist other policies that are able to guarantee the same worst-case performance (e.g. drop each other frame), the Greedy Policy, which tries to optimize the QoS each time a frame is discarded, is a natural one. To establish the competitive ratio of the Greedy Policy we introduce a measure of the optimal maximal distance between discarded frames. Then we derive the cost of the Greedy Policy using this
measure. In particular we demonstrate that the distance between two discarded frames using the Greedy Policy is at least half the measure minus one, and show that it is almost optimal distance for a competitive online policy (up to a factor of 2). First we need some auxiliary lemmas.

Observation 1 When the number of frames lost in a time interval of \( s \) time units is \( k \) the optimal maximal distance satisfies \( d_{\text{OPT}}(k, s) = \left\lfloor \frac{M+k}{k-1} \right\rfloor \).

The observation holds since during \( s \) time units at most \( \frac{M+k}{k-1} \) frames could be accepted. By the monotonicity and the convexity properties of \( \phi_1 \), the cost function is minimized when the discarded frames are equally distributed, i.e., when the minimal distance between two discarded frames is maximized. A simple combinatorial argument shows that the maximum is achieved when discarded frames divide the \( \frac{M+s}{k-1} \) accepted frames into \( \frac{k-1}{k} \) balanced parts. The following lemma compares the performance of the Greedy Policy to \( d_{\text{OPT}} \).

Lemma 1 When frames are scheduled according to the Greedy Policy and the size of the burst is bounded by \( \frac{M+k}{k} \) the minimal distance between two adjacent discarded frames is at least \( d_{\text{OPT}}(k, 0)/2 - 1 \).

Proof. Suppose by way of contradiction that the Greedy Policy discards a frame violating the condition of the lemma. Let the set of the discarded frames succeeding the latest accepted frame currently in the buffer be \( L \) and let the cardinality of \( L \) be \( m \leq k - 1 \). We assume that frames in the \( L \) are enumerated starting from the end of the sequence. Let \( d_0 \) denote the number of accepted frames in the buffer preceding the most recent discarded frame in \( L \), \( d_i \) denote the distance between the \( i \)'th and \( i + 1 \)'th discarded frames in \( L \), and \( d_m \) denote the number of accepted frames in the buffer succeeding the latest discarded frame in \( L \) or the distance between the latest discarded frame in \( L \) and the previous discarded frame if any.

The Greedy Policy always tries to maximize the resulting minimal distance between two discarded frames by definition of well-behaved cost function. If the condition of the lemma is violated then the following holds. The distance between any pair of adjacent discarded frames in \( L \) is at most \( d_{\text{OPT}}(k, 0) - 2 \). If it is not the case the Greedy Policy would have been able to discard a frame without having the minimal distance falling below \( d_{\text{OPT}}(k, 0)/2 - 1 \). Observe that the buffer is full and each of the frames currently in the buffer takes part in exactly one of the above distances. Thus the sum of distances bounds \( M \), and hence,

\[ M \leq \sum_{i=0}^{m} d_i \leq d_0 + (m-1)(d_{\text{OPT}}(k, 0) - 2) + d_m \leq (k-1)(d_{\text{OPT}}(k, 0) - 2). \]

The last inequality follows from the fact that if there exists a discarded frame not in \( L \) then \( d_m \leq d_{\text{OPT}}(m, 0) - 2 \) and \( m \leq k - 2 \), otherwise \( d_m \leq d_{\text{OPT}}(m, 0)/2 - 1 \) and \( m \leq k - 1 \). By this way we obtain that
which contradicts with Observation 1.

In the next theorem we show that the Greedy Policy is $L/U$-competitive. Notice that the Greedy Policy schedules a maximum number of frames.

**Theorem 1** For any sequence generated by a $(\sigma, p)$-source with $\sigma \leq \frac{3}{4}M$, the competitive ratio of the Greedy Policy is at least $L/U$.

**Proof.** Notice that $OPT$ discards exactly the same number of frames as Greedy Policy does. Observation 1 and Lemma 1 imply that the minimal distance between two adjacent discarded frames is at least 1. So the Greedy Policy incurs cost of at most $U$ per frame since $OPT$ has to discards frames with cost of at least $L$. □

Next we show that there exists a sequence of frames such that no online policy could improve the $d_{OPT}(M/4, 0)$ distance measure while $OPT$ is able to discard frames arbitrarily far apart. The intuition behind the proof is that no online policy can discard frames at time of underflow without being 0-competitive. That means that a competitive online policy should be able to receive a burst of maximum size simultaneously, while $OPT$ could smoothly distribute the discarded frames throughout the sequence.

**Theorem 2** There exists an input sequence generated by a $(\sigma, p)$-source with $\sigma \leq \frac{3}{4}M$ so that any online policy discards frames at distance of at most $d_{OPT}(M/4, 0)$ while in $OPT$ the distance between any two discarded frames is arbitrarily large.

**Proof.** Suppose that frames are scheduled according to an online policy $A$. We construct a sequence of frames so that on the one hand $OPT$ either does not discard any frame or the distance between two discarded frames is arbitrarily large and on the other hand $A$ either discards at least one frame or discards $M/4$ frames that are very close.

Consider the following scenario. At time $t_s = 0$ the buffer is empty and $M$ frames arrive. Then during sufficiently long scheduling period till time $t_f$ one frame arrives every time unit. The online policy $A$ cannot discard any of these frames without being 0-competitive.

A burst of $M/4$ frames arrives at the end of the period (at time $t_f$). Note that since $A$ did not discard any frame its buffer is full. Henceforth $A$ necessarily discards $M/4$ frames when the burst arrives at time $t_f$. According to Observation 1 the optimal distance between two discarded frames is $d_{OPT}(M/4, 0)$. However, $OPT$ could have evenly distributed the $M/4$ discarded frames until time $t_f$ and accept all the burst of $M/4$ frames. Therefore the distance between two discarded frames is $d_{OPT}(M/4, t_f - t_s)$, which can be made arbitrarily large. □

The following is an immediate consequence of Theorem 2.

**Corollary 1** The competitive ratio of any online policy is at most $L/U$. 

4.2 Large Burst Size

We consider video streams generated by a $(\sigma, p)$-source with $\sigma > \frac{5}{4} M$. This is an interesting case for bursty VBR video streams. Notice that now the Greedy Policy may be forced to discard consecutive frames. In a similar spirit to the distance between discarded frames, we obtain bounds on the lengths of lost blocks. The problem here is that the dependence of a cost of such a sequence on its length is not linear but quadratic. This means that the cost of loss might increase drastically when $\sigma$ increases.

Nevertheless, it turns out that in this case $OPT$ also suffers large loss. We establish that the competitive ratio of the Greedy Policy is $\min(\frac{U'}{11}, \frac{U'}{15})$. In order to show this we introduce an measure of the optimal minimal length of a lost block. Then we derive the cost of the loss of the Greedy Policy and $OPT$ in terms of this measure. Moreover, we show that the length of a lost block produced by the Greedy Policy is at most factor of 4 larger than the lower bound for $OPT$. Before we prove the main theorem we show a few lemmas. The following observation states the value of $l_{OPT}$. First we introduce the following definition.

**Definition 10** Let the lost set of a policy at time $t$ be the set of discarded frames that arrived after $t - M - 1$.

**Observation 2** When the number of frames lost throughout a time interval of $s$ time units is $k$, the maximal length of a lost block satisfies $l_{OPT}(k, s) = \left\lceil \frac{k}{M + s + 1} \right\rceil$.

The observation holds since during $s$ time units at most $M + s$ frames could be accepted. A simple combinatorial argument shows that any schedule will have a lost block of size at least $\left\lceil \frac{k}{M + s + 1} \right\rceil$. The proof of the following lemma is omitted.

**Lemma 2** When frames are scheduled according to the Greedy Policy and the size of the current lost set is $k$ the Greedy Policy discards at least $k$ frames from the current lost set and pays cost of at least $L' \cdot \frac{k}{2} k$.

For simplicity we assume in the sequel that $l_{OPT}(k, s) = \frac{k}{M + s + 1}$.

**Observation 3** The cost of $k$ discarded frames divided into lost blocks of length 1 under a well-behaved function is bounded from below by $L' \cdot \frac{k}{2} k$ and bounded from above by $U' \cdot \frac{k}{2} k$.

Next lemma shows that for large bursts $OPT$ unavoidably pays the cost of producing sizable lost blocks.

**Lemma 3** When frames are scheduled according to the Greedy Policy and the size of the current lost set is $k > M$ $OPT$ discards at least $k - M$ frames from the current lost set and pays cost of at least $L'(l_{OPT}(k - M, M) + 1)(k - M)/2$. 
Proof. The current lost set contains frames that arrived during the last $M$ time units. Observe that the number of frames accepted in this time interval by the Greedy Policy and by $OPT$ may differ by at most $M$ frames, i.e. $OPT$ is able to accept $M$ additional frames if its buffer is empty at the beginning of the interval. The lemma follows by Observation 2 and Observation 3.

The next proof uses a technique of “$k$-matching” between loss of the Greedy Policy and $OPT$. In $k$-matching the loss of the online policy are divided into disjoint sets that are matched to sets of frames lost by $OPT$ so that each frame lost by $OPT$ appears in at most $k$ such sets. The loss of the Greedy Policy are matched to the loss of $OPT$ either by 1-matching or by 3-matching. Evidently, the competitive ratio of the Greedy Policy is at least the minimum of the minimum ratio of 1-matched sets and one third of the minimum ratio among 3-matched sets.

Theorem 3 For any sequence generated by a $(\sigma, p)$-source with $\sigma > \frac{5}{4}M$, the competitive ratio of the Greedy Policy is at least $\min(\frac{L}{4M^2}, \frac{L}{15M})$.

Proof. We divide the schedule of the Greedy Policy into intervals of length $M$ and consider lost sets at the last time moment of every interval. Note that each discarded frame belongs to exactly one of these sets. These lost sets are matched to the loss of $OPT$ either by 1-matching or by 3-matching. First we identify and match the lost sets of the Greedy Policy that participate in 3-matching. Having finished with 3-matching, the remaining lost sets participating in 1-matching are arbitrarily matched to sets of the same cardinality formed from the left unmatched frames lost by $OPT$. Notice that both Greedy Policy and $OPT$ lose the same number of frames. Thus there remains a sufficient number of free frames lost by $OPT$ for 1-matching provided that 3-matched online lost sets have cardinality smaller than their offline counterparts.

We bound the cost of a lost set of the processed interval by determining the maximal length of a lost block frames from the interval’s lost set could belong to. Suppose that we process the $i$’th interval $[Mi, M(i+1)]$. Let $k$ be the cardinality of the lost set of the interval (i.e. at time $M(i+1) - 1$) and let $t_{\text{max}}$ be the time moment within the interval $M(i - 1) \leq t_{\text{max}} < M(i + 1)$ in which the cardinality $k_{\text{max}}$ of the lost set is maximal. Clearly, the maximal length of a lost block is the measure of the Lemma 2 at time $t_{\text{max}}$, that is $2l_{OPT}(k_{\text{max}}, 0) + 1$.

Now the value of $k_{\text{max}}$ determines the kind of matching in the following way.

1. If $k_{\text{max}} > 10M$ then the lost set of the interval takes part in 3-matching.

Next we show how to find a matching of the interval lost set to a set of frames lost by $OPT$, whose cost constitutes at least $\frac{L}{115M}$ fraction of its cost. We match the lost set to a set of $k_{\text{max}} - M$ frames that were necessarily lost by $OPT$ during time interval $[t_{\text{max}} - M - 1, t_{\text{max}}]$. This is indeed a 3-matching since each of the offline lost frames could participate in at most three such sets, that is the matched set of the interval and the corresponding matched sets of the neighboring intervals.

The Lemma 3 implies that the cost incurred by $OPT$ is at least
At the same time, by Observation 3, the cost incurred by the Greedy Policy is at most
\[ U'(l_{OPT}(k_{max}, 0) + 1)k \leq U'(l_{OPT}(k_{max}, 0) + 1)k_{max}. \]

Hence, we can bound the ratio between the cost of the offline and the corresponding online lost sets by \( \frac{U'}{U'} \), since \( k_{max} > 10M \).

(2) If \( k_{max} \leq 10M \) then the lost set of the interval takes part in 1-matching. By Observation 2 the optimal maximal length of a lost block is at most 10. Applying Lemma 2 we obtain that the maximal length of a lost block of discarded frames for the Greedy Policy is at most 21. Therefore according to Observation 3 the cost of the lost set is upper bounded by \( 11k \cdot U' \). The ratio is kept above \( \frac{L}{10U'} \) since each lost frame has cost at least \( L \).

The competitive ratio of the Greedy Policy is the minimum of the ratio of 1-matched sets and one third the ratio of 3-matched sets, which establishes the theorem.

Next we show that the length of a lost block created by the Greedy Policy is at most four times the optimal length of a lost block of \( OPT \) plus three.

**Lemma 4** For any input sequence \( S \) the length of the maximal lost block satisfies \( l_{GREEDY}(S) \leq 4 \cdot l_{OPT}(S) + 3 \).

**Proof.** By Lemma 2 the maximal length of a lost block produced by the Greedy Policy is at most \( 2l_{OPT}(k, 0) + 1 \) when the size of the lost set is \( k \). At the same time by Lemma 3 the optimal length of a lost block in this case is at least \( l_{OPT}(k - M, M) \). The theorem follows since,
\[ 2l_{OPT}(k, 0) + 1 \leq 4l_{OPT}(k - M, M) + 3. \]

\( \square \)

We conclude with the following general theorem.

**Theorem 4** For any sequence of frames, the competitive ratio of the Greedy Policy is at least \( \min\left\{ \frac{L}{U'}, \frac{L}{10U'}, \frac{L'}{10U'} \right\} \).

## 5 Conclusion

In this work we study competitive online buffering policies for video transmission across internetwork with leaky bucket constraints. We consider video encoding schemes with independent frames, such as JPEG and its derivatives. To measure the QoS we define a well-behaved cost function reflecting the playback discontinuity at the client. For moderate and large burst parameters we derive the
competitive ratio of the Greedy Policy. In addition to the global QoS function we consider local metrics, such as the minimal distance between two discarded frames and maximal length of a sequence of consecutive discarded frames. The Greedy Policy is shown to be competitive with regards to these metrics as well.

The proposed policy may be used for managing current Internet routers that wish to provide QoS. A distributed implementation of the Greedy Policy may be easily deployed and operate at very high speeds. Interesting future directions include studying more sophisticated cost functions and performing simulations in which performance of online dropping policies is estimated more properly using global QoS function.

References


**Alexander Kesselman**  Department of Computer Science, Tel-Aviv University, *email: alx@math.tau.ac.il*.

**Yishay Mansour**  Department of Computer Science, Tel-Aviv University, *email: mansour@math.tau.ac.il*. 
On Immunity and Catastrophic Indices of Graphs*

RASTISLAV KRÁLOVIČ
Comenius University, Slovakia

PETE R RUŽIČKA
Comenius University, Slovakia

Abstract

Immunity index of a graph is the least integer $c_1$ such that each configuration of size $c_1$ is immune. Catastrophic index of a graph is the least integer $c_2$ such that each configuration of size $c_2$ is catastrophic.

This paper contains the first systematic study of immunity indices on a variety of interconnection networks and their distance from catastrophic indices.

Keywords

distributed computing, fault tolerance, interconnection networks

1 Introduction

We consider the following coloring game played on a simple connected graph. The game proceeds in synchronous rounds and uses a set of colors \{black, white\}. Initially, vertices are colored black or white. At each round, each white vertex simultaneously recolors itself by the color of the simple majority of its neighbours. A set of vertices $M$ is said to be a dynamic monopoly (shortly dynamo) if starting with only the vertices of $M$ colored black the game eventually reaches the all-black configuration. Such a dynamo is called irreversible to stress the fact that initially black vertices are permanently black (i.e. they cannot be recolored to white). The importance of this game follows from the fact that it models faulty behaviour of point-to-point systems based on majority voting. In this case, dynamos correspond to sets of initial faults that cause the entire system to fail.

An immune subgraph of a graph is a subgraph for which there does not exist a computation starting from some initial configuration with all vertices of the immune subgraph white and ending in all-black configuration. In this case, immune

* Supported in part by grant from VEGA 1/7155/20.
subgraphs correspond to sets of correct nodes that prevent the entire system from falling.

Dynamic monopolies have been introduced in [L+93] and since that time they have been intensively studied in the literature. The main line of the previous research has been oriented on the size of dynamos for various topologies (see e.g. survey paper [P01]). Recently, the systematic study of dynamos with respect to both the size and the time (i.e. how many rounds are needed to reach the all-black configuration) was presented in [F+00].

Immunity subgraphs has been introduced in [P96], where the connection between immunity and expansion has been studied.

In the “dynamo size problem” we ask whether there is a catastrophic configuration of certain size. The natural question to ask is: For which sizes are all configurations catastrophic. This is formalized in the notion of catastrophic index. In analogy, we can introduce the notion of immunity index.

We define immunity index $c_1$ of a graph $G$ as the least integer such that each configuration of size $c_1$ is immune. Clearly, $c_1 = N - m + 1$, where $m$ is the size of a smallest dynamo of $N$-vertex graph $G$.

We also introduce catastrophic index $c_2$ of a graph $G$ as the least integer such that each configuration of size $c_2$ is catastrophic. Later on it is proved that $c_2 = N - d + 1$, where $d$ is the size of a minimum immune subgraph of $N$-vertex graph $G$.

In this paper we propose to study catastrophic indices of graphs, i.e. to study minimal immune subgraphs in detail. We are interested in the following questions:

(1) What is the size of smallest immune subgraphs in a given graph?

(2) Given a graph, can we partition it into (minimal) immune subgraphs? (The maximal number of disjoint immune subgraphs in such a partition is called the immunity degree of $G$.)

## 2 Basic Notions

Let $G = (V, E)$ be a simple connected graph and $N = |V|$. Assume that the vertices of the graph $G$ are colored (black or white). By a configuration on $G$ we mean a partition of $V$ to the set of black and the set of white vertices. For simplicity, a configuration will be referred to as the set of its black vertices. By the size of a configuration we mean the number of its black vertices.

The final configuration (all-black configuration) of $G$ is the configuration with all vertices black. A catastrophic configuration is a configuration from which there is a synchronous computation (i.e. each round of which is performed by means of a simple majority rule applied to each white vertex of $G$ simultaneously) leading to the all-black configuration. Such a computation is called a complete computation.
By time complexity of a complete computation we mean the number of its rounds. A dynamic monopoly (shortly dynamo) is a set of black nodes whose corresponding configuration is catastrophic. By $t$-time computation we mean a complete computation having $t$ rounds. By $t$-time dynamo we mean a dynamo having complete computation with $t$ rounds. If not mentioned explicitly, by a dynamo we mean an irreversible dynamo using simple majority rule. By a minimal dynamo we mean a dynamo of minimal size.

Given a dynamo $M$, let $M_t$ denote the set of black-colored vertices after round $t$ (with $M_0 = M$). A dynamo $M$ is monotone if $M_t \subseteq M_{t+1}$ for every $t \geq 0$. It is obvious that irreversible dynamos are monotone.

An immune subgraph $G'$ of $G$ is a subgraph for which there does not exist a complete computation on $G$ starting from some initial configuration with all vertices from $G'$ white. This means that a subgraph $G' \subseteq G$ is immune if and only if for each vertex $v \in V'$ it holds $\deg_{G'}(v) = |\text{Neigh}_{G'}(v) \cap V'| > \frac{\deg_{G}(v)}{2}$. By $t$-immune subgraph we mean an immune subgraph $G' \subseteq G$ such that there does not exist a $t$-time computation on $G$ starting from a configuration with all vertices from $G'$ white. An immune configuration is a set of vertices of some immune subgraph of $G$.

By catastrophic index of a graph $G$ we mean the least integer $c$ such that each configuration with at least $c$ black vertices is catastrophic. By immunity index of a graph $G$ we mean the least integer $c$ such that each configuration of size $c$ is immune. The immunity degree is the maximal number of disjoint immune subgraphs in $G$.

3 Catastrophic indices and smallest immune subgraphs

Recently, there has been intensive study on the size of dynamos for various topologies. Tight size bounds have been obtained for complete trees, rings [F+00], tori [F+98], butterflies [LPS99, F+00], cube-connected cycles [LPS99] and shuffle-exchange graphs [F+00]. There are still asymptotical differences between the upper and lower bounds on the size of minimal dynamos for wrap-around butterflies, De Bruijn graphs, hypercubes and star graphs and the open question is to tighten these differences.

In the problem of “dynamo size” we ask about the existence of catastrophic configuration of certain size. A natural question is to ask, for which sizes all configurations are catastrophic. Next property relates the catastrophic index of graphs to the size of their smallest immune subgraphs.

**Theorem 1** For the catastrophic index $c$ of $G$ it holds $c = N - m + 1$, where $N$ is the number of vertices in $G$ and $m$ is the size of the smallest immune subgraph of $G$. 
Proof. Let $G'$ be the minimal immune subgraph of $G$ with $m$ vertices. Clearly, for each $c^d \leq N - m$ there is a configuration with $c^d$ black vertices which is not a dynamo (the configuration with $G'$ white), hence $c > N - m$.

Consider a configuration with $N - m + 1$ black vertices. Suppose, for the sake of contradiction, that it is not a dynamo. Then the computation stops in a configuration with at most $m - 1$ white vertices. Each of the white vertices has more than half of its neighbours white, hence there is an immune subgraph of size $m - 1$. □

As a consequence of the previous theorem, several interesting estimates can be obtained.

Observation 1 For each $N > 3$ there exists an $N$-vertex graph with the catastrophic index at least $N - 3$.

Proof. Consider a graph $K_4$ sharing one vertex with an arbitrary $(N - 4)$-vertex graph. Then $K_3$ is an immune subgraph. □

Observation 2 The catastrophic index of an $r$-regular graph $G$ is at most $N - \lfloor \frac{r}{2} \rfloor + 1$, where $N$ is the number of vertices in $G$.

Proof. Follows from the fact that each immune subgraph of an $r$-regular graph $G$ has at least $\lfloor \frac{r}{2} \rfloor$ vertices. □

4 Size of minimal immune subgraphs

As we have seen in the previous subsection, the crucial point in determining the catastrophic indices of graphs is to determine the size of the smallest immune subgraphs.

The first question to ask is about the complexity of the following decision problem: Given a graph $G$ and an integer $k$, is there an immune subgraph of size $k$? In the next theorem we show that this problem is $NP$-complete.

Theorem 2 The following decision problem is $NP$-complete: Given a graph $G$ and an integer $k$, is there an immune subgraph of size at most $k$?

Proof. Reduction from ExactCoverBy3-Sets (EC3S). The instance of EC3S consists of a set $A = \{a_1, \ldots, a_{3m}\}$ and a system of subsets of $A$: $S = \{S_1, \ldots, S_n\}$ such that $S_i \subseteq A$, $|S_i| = 3$. The instance is positive, if there exists a set $C = \{C_1, \ldots, C_m\}$ such that $C_i \in S$ and $\bigcup_{i=1}^{m} C_i = A$.

Construct a graph $G = (V, E)$ as follows. Start with a complete bipartite graph with bipartition $\{v_1, \ldots, v_n\}$ and $\{a_1, \ldots, a_{3m}\}$. For each set $S_i$, add vertices $\{b_{i1}, b_{i2}, b_{i3}\}$ with degree two, such that $b_{ij}$ is connected with $v_i$ and $a_{ij}$, where
Moreover, let us call a stop-tree the graph consisting of a root and 15 leaves. For each vertex \( v_i \) add 3 stop-trees with roots connected to it, and for each vertex \( a_i \) add \( m \) stop-trees with roots connected to it.

It holds that \( G \) has an immune subgraph of size at most \( 7m \) if and only if the corresponding instance of EC3S is positive.

Clearly if the EC3S instance is positive, consider the subgraph \( I \subseteq G \) consisting of all vertices \( a_i \), vertices \( v_j \) such that \( S_j \in C \), and vertices \( b_{kl} \) such that \( v_k \in I \). I is immune and has size \( 7m \).

On the other hand, if there exist an immune subgraph of size \( 7m \), it contains no vertices from stop-trees. Moreover, it contains all vertices \( a_i \). From the size requirements it follows that the vertices \( v_i \) contained in \( I \) form a positive instance of EC3S.

Another question to ask is: what is the size of minimal immune subgraphs in well-known interconnection networks? We give tight bounds for tori, cube-connected cycles and hypercubes, and upper bounds for butterflies and star networks.

In the proof of the following theorem we need this useful lemma.

**Lemma 1** [Ch+88] Edge isoperimetric inequality for a \( d \)-dimensional hypercube \( Q_d \) is

\[
\partial_{Q_d}(m) \geq m(d - \log_2 m)
\]

where \( \partial_{Q_d}(m) \) denotes the size of the minimal edge boundary of an \( m \) vertex subset of vertices of \( G \).

**Theorem 3** The size of a minimal immune subgraph in

1. a tori \( T_{d_1 \times d_2} \) is \( 2 \cdot \min(d_1, d_2) \);
2. a hypercube \( Q_d \) is \( 2^{\lceil \frac{d+1}{2} \rceil} \);
3. a butterfly \( BF_d \) is at most \( 3 \cdot 2^{\lfloor (d+1)/2 \rfloor} (4 - 2^{d \mod 2}) - 8 \);
4. a cube connected cycles \( CCC_d \), \( d \geq 8 \), is 8;
5. a star \( S_d \) is at most \( \lceil \frac{d+1}{2} \rceil \)!
Proof.

1. W.l.o.g. let \( d_1 < d_2 \). Consider an orientation such that the \textit{up-down} direction is in the dimension of \( d_1 \). First we show that if \( S \) is a minimal immune subgraph, then \( S \) contains one full column of vertices. From that, the result follows: every vertex in \( S \) has at least three neighbours in \( S \) so it is possible to assign to each vertex in the row of \( d_1 \) vertices a distinct vertex in \( S \). To show that there is one row in \( S \) first note that two rows form an immune subgraph of \( 2d_1 \) vertices. Consider an arbitrary immune subgraph \( S \). As every vertex in \( S \) has at least three neighbours in \( S \) there is a closed path \( p \) consisting of \textit{down} and \textit{right} edges only. Suppose there is no such column. Then there are two possibilities for \( p \). If \( p \) is a complete row then by argument similar to above \( S \) contains \( 2d_2 \) vertices and is not minimal. If \( p \) is not a complete row then it must cross all four “borders” hence the length is at least \( d_1 + d_2 \) and \( S \) is not minimal.

2. Let \( S \) be an immune subgraph of \( Q_d, |S| = m \). For each vertex \( v \in S \) there are \( \left\lfloor \frac{d+1}{2} \right\rfloor \) neighbours in \( S \), so there are at most \( m \left( \left\lfloor \frac{d+1}{2} \right\rfloor \right) \) edges outgoing from \( S \). Using Lemma 1 we get \( m \left( \left\lfloor \frac{d+1}{2} \right\rfloor \right) \geq m(d - \log_2 m) \) which implies the claim.

3. Let \( k = \left\lfloor \frac{d}{2} \right\rfloor \). Consider vertices \((0^d, k)\) and \((0^k1^{d-k}, k)\) together with complete binary trees connecting each of them to the set \( \{0^k\alpha, d+1\} \ | \ \alpha \in \{0, 1\}^{d-k} \} \) and vertices \((1^d, k)\), and \((1^k0^{d-k}, k)\) together with complete binary trees connecting each of them to the set \( \{1^k\alpha, d+1\} \ | \ \alpha \in \{0, 1\}^{d-k} \} \).

Similarly consider vertices \((0^d, k-1)\) and \((0^k1^{d-k}, k-1)\) together with complete binary trees connecting each of them to the set \( \{0^k\alpha, 0\} \ | \ \alpha \in \{0, 1\}^{d-k} \} \) and vertices \((1^d, k-1)\), and \((1^k0^{d-k}, k-1)\) together with complete binary trees connecting each of them to the set \( \{1^k\alpha, 0\} \ | \ \alpha \in \{0, 1\}^{d-k} \} \).

This subgraph is immune and contains four trees of height \( \lfloor (d + 1)/2 \rfloor \) and four trees of height \( \lfloor (d + 1)/2 \rfloor \), where each leaf is in exactly two trees. Thus the number of vertices is \( 3 \cdot 2^{(d+1)/2} \lfloor (d + 1)/2 \rfloor (4 - 2^d \mod 2) - 8 \).

4. CCC is a cubic graph, hence its minimal immune subgraph is the shortest cycle.

5. Star \( S_d \) can be decomposed into \( d \) substars \( S_{d-1} \) by fixing each different symbol in \( \{1, 2, \ldots, d\} \) in one particular position \( 2 \) to \( d \). If we fix a specific symbol in the last position we observe that there are \( (d - 1)! \) vertices (i.e. an \( S_{d-1} \)) for every one of the \( d \) symbols. Thus, the vertices of the \( S_d \) can be partitioned into \( d \) groups, each containing \( (d - 1)! \) vertices and each being isomorphic to \( S_{d-1} \). If this decomposition is recursively applied to the resulting substars, \( S_d \) can be decomposed into \( \frac{d!}{k!} \) substars \( S_k, 1 \leq k \leq d - 1 \).

\( \square \)

Note that following [F+00] it holds for the case of star graph that \( m \geq \frac{N}{d+1} \), where \( m \) is the size of a minimal dynamo on \( S_d \). We are aware of only trivial upper bound on the size of a minimal dynamo in the form \( N/2 \) for \( S_d \).
5 Decomposition to immune subgraphs

In the previous subsection we have concentrated on the problem of determining the size of minimal immune subgraphs. To recall, our motivation to study the minimal immune subgraphs was that they correspond to the failure patterns that do not cause the entire system to fail. Another interesting aspect of this problem is to know how many disjoint immune subgraphs are there. This, in certain sense expresses the immunity degree of the entire distributed system.

In this subsection we give characterization results of the following type: Given a graph $G$, is there a unique (up to some symmetries) decomposition to minimal immune subgraphs? We give tight immunity degree values for tori, cube-connected cycles and hypercubes, and give lower bounds for butterflies and star graphs.

**Theorem 4** There is a decomposition of

1. a tori $T_{d_1 \times d_2}$, $d_1 \leq d_2$, $d_2$-even, to minimal immune subgraphs;
2. a butterfly $BF_d$ to immune subgraphs;
3. a cube connected cycles $CCC_d$, $d$-even, to minimal immune subgraphs;
4. a hypercube $Q_d$ to minimal immune subgraphs;
5. a star $S_d$ to immune subgraphs;

**Proof.** The part (1) is trivial, (2) follows from [LPS99], (4) and (5) follow from the recursive structure of these graphs and the previous theorem.

For the part (3) consider the faces of the original hypercube (i.e. cycles of length 8 in CCC) corresponding to dimensions $(2i, 2i+1)$ for $0 \leq i \leq d/2$.

As a consequence, we obtain exact immunity degree values for certain topologies. The immunity degree of a tori $T_{d_1 \times d_2}$ is $\left\lfloor \frac{d_2}{2} \right\rfloor$ ($d_1 \leq d_2$). The immunity degree of $CCC_d$, $d$-even, $d \geq 8$, is $d \cdot 2^{d-3}$. The immunity degree of $Q_d$ is $2^{\frac{d+1}{2}}$. We also have lower bounds on immunity degrees for other topologies. Immunity degree of $BF_d$ is at least $2^{\left\lfloor \frac{d+1}{2} \right\rfloor}$ and of $S_d$ is at least $d \cdot (d-1) \cdot \cdots \cdot \left( \left\lfloor \frac{d+1}{2} \right\rfloor + 1 \right)$.

Immunity degree of a graph gives a lower bound on the size of a minimal dynamo in this graph. For several topologies we obtained rather weak lower bounds. However, for $CCC_d$ (and also wrapped butterflies – but the result is not mentioned here) we get the best known lower bounds, previously established in [LPS99].

6 Minimal $t$–immune subgraphs

Up to now, we have concentrated on the size of dynamos and immune subgraphs. The size of dynamo is clearly a crucial parameter: a large size implies a less
likely occurrence. Thus, a system in which the smallest dynamo is large has a high degree of fault-tolerance. However, the size is not the only interesting aspect of the quality of dynamos. In particular, the time needed for a dynamo to converge into an all-black configuration is a very important characteristic, not only from a combinatorial but also from a practical point of view. If a catastrophic set of faults has a slow evolution, its presence might be more easily detected; and on the other hand, a fast dynamo is inherently more dangerous for the system.

The systematic study of dynamos with respect to both the size and the time has been done in [F+00] for various models and topologies. Tight tradeoffs between the size and the time has been presented for rings, complete \(d\)-ary trees, tori, wrap-around butterflies, cube-connected cycles and hypercubes. The question remains to determine the size of minimal \(t\)-time irreversible dynamos for De Bruijn graphs.

The study of minimal \(t\)-immune subgraphs is important, mainly for their close relationship to the problem of determining \(t\)-catastrophic indices of networks.

We give tight bounds on the size of minimal \(t\)-time immune subgraphs for rings, tori and hypercubes and upper bounds for butterflies.

We define \(t\)-immunity index of a graph \(G\) as the smallest integer \(c\) such that each configuration of size \(c\) is \(t\)-immune. Clearly, \(c = N - m + 1\) where \(m\) is the size of a minimal \(t\)-dynamo of \(N\)-vertex graph \(G\). We also define \(t\)-catastrophic index of a graph \(G\) as the smallest integer \(c\) such that each configuration of size \(c\) is \(t\)-catastrophic. Clearly, \(c = N - d + 1\) where \(d\) is the size of a minimum \(t\)-immune subgraph of \(N\)-vertex graph \(G\).

Let \(B_G(v, r)\) denote a ball with radius \(r\) in graph \(G\) centered in vertex \(v\) (i.e. the set of vertices in \(G\) with distance at most \(r\) from \(v\)). Note, that while every ball with radius \(t\) of an immune subgraph is \(t\)-immune, it may be the case that a \(t\)-immune subgraph is not contained in any minimal immune subgraph. As an example consider a graph \(G\) consisting of two cliques of size \(2^t\) and one hypercube of dimension \(t\), each connected to a root vertex \(v\) with paths of length \(t\). A ball centered in \(v\) with radius \(t\) is \(t\)-immune subgraph of \(G\) with \(1 + 3t\) vertices. We claim that each minimal immune subgraph of \(G\) is fully contained in the hypercube. First note that any \(t/2\)-dimensional subcube (not containing the vertex on the path towards \(v\)) is immune subgraph of \(G\) with \(2^{t/2}\) vertices. On the other hand every immune subgraph containing \(v\) contains at least one vertex from a \(2^{t/2}\)-clique and hence its size is at least than \(2^{t+1}\).

**Theorem 5** Minimal \(t\)-immune subgraphs of

1. \(N\)-vertex ring \(R_N\) are of size \(2t + 1\) (1 \(\leq\) \(t\) \(\leq\) \(\lfloor \frac{N-1}{2} \rfloor\));

2. square \(n \times n\) tori are of size \(4t\) (1 \(\leq\) \(t\) \(\leq\) \(\lfloor \frac{N-1}{2} \rfloor\));

3. \(d\)-dimensional butterfly \(BF_d\) are of size at most \(3 \cdot 2^t - 2\) for (1 \(\leq\) \(t\) \(\leq\) \(\lfloor \frac{d-1}{2} \rfloor\));

4. \(d\)-dimensional hypercube \(Q_d\) are of size \(\sum_{i=0}^{t} (\binom{d+1/2}{i})\);
Proof. (1) The minimal immune subgraph of a ring $R_N$ is of size $N$. For $t$, $1 \leq t \leq \left\lceil \frac{N-1}{2} \right\rceil$, any consecutive subset of $2t+1$ vertices forms a $t$-immune subgraph. As every computation starting from a configuration with at most $2t$ white vertices has time at most $t-1$, this $t$-immune subgraph is minimal.

(2) Consider two neighbouring columns in tori $T_{n \times n}$. It is obvious that they form minimal immune subgraph $T$ of $T_{n \times n}$. As $B_T(v,t)$ has size $4t$ we have the upper bound. For the lower bound note that every configuration with more than four and less than $2n-1$ vertices has at least four “corners” (by a corner we mean a black vertex with at most two black neighbours) and each configuration with four or $2n-1$ vertices has at least three corners. Consider a computation starting from a configuration with fewer that $4t$ black vertices. The lower bound comes from the fact that as $t \leq \left\lceil \frac{N-1}{2} \right\rceil$ in each step at least four vertices will be recolored black.

(3) Let $k = \left\lceil \frac{d}{2} \right\rceil$. Consider $B_T((0 \ldots 0,k),t)$ where $T$ is the immune subgraph from Theorem 3. For $t < k$ it consists of two complete binary trees of heights $t$ and $t-1$ thus having $2^{t+1} - 1 + 2t - 1$ vertices.

(4) Let $k = \lfloor d + 1/2 \rfloor$. From Theorem 3 it follows that the minimal immune subgraph is the $k$-dimensional subcube. The upper bound is obtained by $B_{Q_k}(v,t)$ for any vertex $v$. For the lower bound we prove the following: for any $t$-immune subgraph $G$ there is a vertex $v$ such that there are at least $\binom{k}{i}$ vertices in $G$ that have distance $i$ from $v$. Consider a computation starting from a configuration with only vertices from $G$ white. Clearly, as $G$ is $t$-immune there must be a vertex $v$ that is white after $t$ steps. Let $S_0, S_1, \ldots, S_t$ be a sequence of sets $\cup_{i=0}^t S_i \subseteq G$, where $S_{i-1}$ are vertices that are recolored after $i-1$ steps ($v \in S_0$). Each $w \in S_i$, $0 \leq i < t$ must have at least $k$ neighbours in $\cup_{j=0}^{i-1} S_j$. Moreover, $w$ has at most distance $i$ from $v$, hence at most $i$ neighbours in $\cup_{j=0}^{i-1} S_j$ with distance $i-1$ from $v$. It follows that $v$ has at least $k-i$ neighbours $w$ in $S_{i+1}$ with distance $i+1$ from $v$. Each of these can have at most $t+1$ neighbours in $S_i$. We use induction to argue that there is at least $\binom{k}{i}$ vertices in $S_i$ and hence at least $\binom{k}{i+1}$ vertices in $S_{i+1}$. □

We have only trivial upper bound on the size of minimal $t$-immune subgraph in star graphs. The question remains to determine exact bounds.

7 $t$–dominating set of minimal immune subgraphs

Decomposition of a network to minimal $t$–immune subgraphs with respect to perfect $t$–dominating sets is important in order to guarantee the fault-tolerant quality of $t$–time simple majority computations in networks.

In the next theorem we give the tight size of minimal $t$-dominating set (i.e. dominating set with radius $t$) of a minimal immune subgraph for a squared tori and an upper bound on the size for butterflies. The question for hypercubes and star graphs seems to be far from trivial.
Theorem 6 Minimal $t$–dominating set of a minimal immune subgraph of

1. squared tori $T_{n\times n}$ is of size $\left\lceil \frac{n}{2} \right\rceil$ ($1 \leq t \leq \left\lceil \frac{n+1}{2} \right\rceil$);

2. BF$_d$ is of size at most $\left\lceil \frac{d+1}{2} \right\rceil \cdot 2\left\lceil \frac{d-1}{2} \right\rceil$, $1 \leq t$, $2t + 1 \leq d + 1$.

Proof. (1) Let $T$ be the minimal immune subgraph of tori $T_{n\times n}$. Consider $B_T(v)$ as in the proof of the previous observation. The size of $B_T(v)$ is $4t$ for $1 \leq t \leq \left\lfloor \frac{n-1}{2} \right\rfloor$. There are $\left\lfloor \frac{n}{2} \right\rfloor$ disjoint $B_T(v)$s in each $T$. The distance of its centers is $2t + 1$. It follows that the set of centers of $B_T(v)$s forms minimal $t$-dominating set of a minimal immune subgraph of $T_{n\times n}$.

(2) Follows from immune subgraphs constructed in [LPS99] and from directly constructed $t$-dominating sets of butterfly columns of size $d + 1$.

In the next observation we present partial results for 1-dominating sets.

Observation 3

1. Minimal 1–dominating set of a smallest immune subgraph of BF$_d$ is of size at most $\left\lceil \frac{d+1}{2} \right\rceil \cdot 2\left\lceil \frac{d-1}{2} \right\rceil$.

2. Minimal perfect 1–dominating set of a smallest immune subgraph of Q$_d$ is of size $2\left\lceil \frac{d-1}{2} \right\rceil$, for $d = 2\left(2^k - 1\right)$.

3. Minimal 1–dominating set of a smallest immune subgraph of S$_d$, $d$–even, is of size at most $\frac{1}{3} \left(\frac{d}{2}\right)!$.

Proof. (1) The consequence of the previous theorem for $t = 1$.

(2) The result appears e.g. in [DR97].

(3) Perfect 1–dominating set of S$_3$ is of size 2. There is a decomposition of S$_{\left\lceil \frac{d+1}{2} \right\rceil}$ into $\left(\frac{d}{2}\right)!$ substars S$_3$, thus the result follows.

8 Distances of immunity and catastrophic indices

In this section we give some notes concerning the relationship between the sizes of immunity and catastrophic indices. First, we show that there are graphs having immune subgraphs of size $\Theta(n)$. The trivial example is a ring $R_n$, but there are also non-constant-degree graphs with this property. As an example of $k$-regular graphs with large minimal immune subgraphs we present the class of Ramanujan graphs [LPS88]: $k$-regular Cayley graphs with the second smallest eigenvalue of their Laplacian matrix bounded by $\lambda \geq k - 2\sqrt{k-1}$. These graphs satisfy a number of extremal combinatorial properties (e.g. large girth). Ramanujan graphs are also the best known explicit expanders.
Theorem 7  Ramanujan graphs have immune subgraphs of size $\Theta(N)$.

Proof.  Let $\lambda$ be the second smallest eigenvalue of Laplacian. Consider a vector $x$ of vertex weights, such that $x_v = 1$ for vertices from the immune subgraph and $x_v = -1$ otherwise. Following [F75] it holds

$$\lambda \leq \frac{N \sum_{(v_i, v_j) \in E} (x_i - x_j)^2}{\sum_{v_i, v_j \in V} (x_i - x_j)^2}$$

Consider a Ramanujan graph of degree $k$. Let $\delta = \left\lfloor \frac{k-1}{2} \right\rfloor$. It follows that $\lambda \leq \frac{N\delta}{N-\delta}$.

As $\lambda \geq k - 2\sqrt{k} - 1$, it follows that

$$|I| \geq N \left( 1 - \frac{k-1}{2k-4\sqrt{k}+1} \right)$$

Another interesting question is to characterize graphs, for which the immunity and catastrophic indices are

- equal (or close);
- of great difference.

We mention some notes on special topologies where immunity and catastrophic indices are close (tori $T_{d_1 \times d_2}$ for $d_2 = 2d_1$) or of great differences (rings).

- Ring $R_N$: immunity index is $c_1 = N$; catastrophic index is $c_2 = 1$; immunity degree is 1.
- Tori $T_{d_1 \times d_2}$: immunity index is $c_1 = d_1 \cdot d_2 - d_2 + 1$; catastrophic index is $c_2 = d_1 \cdot d_2 - 2d_1 + 1$; immunity degree is $\left\lceil \frac{d_1}{2} \right\rceil$.

9 Conclusions

We have initiated the study of immune subgraphs with respect to both the size and time–size complexities. We have shown that to determine whether there is an immune subgraph of given size is $NP$-complete in general graphs. We have derived tight sizes of minimal immune subgraphs for tori, cube-connected cycles and hypercubes and tight time–size tradeoffs for rings, tori and hypercubes. Further results on immunity degrees and $t$-dominating sets of minimal immune subgraphs are also given.

10 Acknowledgements

This paper has benefited from many discussions the authors have had with Shmuel Zaks during his stay at Comenius University.
References


Rastislav Královič is with the Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia. E-mail: kralovic@dcs.fmph.uniba.sk

Peter Ružička is with the Institute of Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia. E-mail: ruzicka@dcs.fmph.uniba.sk
On the Complexity of Path Layouts in Bounded Degree ATM Networks:
A Case Study for Butterfly Networks*

MARTIN MAKÚCH
Comenius University, Bratislava

PETER RUŽIČKA
Comenius University, Bratislava

Abstract
We give an optimal shortest path broadcast layout on butterfly ATM networks of size $N$ with load $L = \Theta(N^{1/H})$ for any hop count $H$.

Keywords
ATM networks, Virtual Path Layouts, Broadcast

1 Introduction
Recent development in fiber optic media offers dramatical changes in the area of digital communication networks. The sharp distinction between computer networks, telephone networks and cable TV networks has been replaced by a unified approach. The new technology is called ATM (Asynchronous Transfer Mode). It allows very reliable transmission together with high bandwidth.

In the routing problem for ATM networks, for certain pairs of nodes the end-to-end communication is done along predefined paths in the network, so called virtual paths. The problem is to design these paths optimally. The smallest number of concatenated paths between two nodes is called the hop count, while the load of a layout is the maximum number of virtual paths along any communication link. The hop count relates to the time needed to establish a connection between two nodes and the load measures the size of routing tables at nodes.

The problem to design virtual path layouts in ATM networks has been intensively studied with respect to the network size $N$, load $L$ and hop count $H$ (for results see representative overview [1]). The general result due to Stacho and Vrňo [5] presents all-to-all virtual path layouts for some constant degree networks.

* Supported in part by grand from VEGA 1/7155/20.
in the form $\mathcal{H} = \Theta \left( \frac{\log N}{\log L} \right)$ for any $L$. This result holds for not necessarily shortest path gossip on networks of $O(\log N)$ bounded diameter and so it can be directly applied also to butterfly networks. However, the lower bound holds even for one-to-all virtual path layouts, that means this result is optimal even for broadcast.

The interesting question is what happens in the reverse case, namely, “given a hop number minimize the load”. The paper suggests that from the theoretical point of view, it is more reasonable (and harder) to study this problem. A good solution to this problem may provide an efficient solution to the converse one, but not vice versa.

In this paper, we present optimal shortest paths broadcast layouts on butterfly networks in the form $L = \Theta \left( \frac{\log N}{\log L} \right)$ for any $\mathcal{H}$. That means, for specific butterfly networks it holds $\mathcal{H} = \Theta \left( \frac{\log N}{\log L-O(1)} \right)$ for any $L$ even for shortest path layouts. The construction can be applied also to some related networks, as wrapped butterflies or cube connected cycles, within the same bound, however by loosing the shortest path property.

There are some questions left open. The main open problem remains whether this reverse result holds also for other topologies of constant degree and $O(\log N)$ bounded diameter. Further, we have restricted just to one-to-all layouts, so the all-to-all case is another possible extension of our results. Here the unresolved problem is whether the all-to-all case have the same asymptotic solution as the one-to-all case, like in the result of Stacho and Vrťo. Finally, a reason of difficulty to analyze the reverse problem is probably because of a strong requirement that any virtual channel to be obtained by concatenating (appropriate) virtual paths should be shortest. The question is what is the impact of the requirement “to be shortest” (or stretch factor to be equal 1) on the generalization to other (larger) class of graphs.

## 2 The Model

We exploit the graph-theoretic model from [1]. The communication network is presented by an undirected graph $G = (V, E)$, where the set of vertices $V$ corresponds to ATM switches, and the set of edges $E$ to physical links between them. Moreover, we have a given set $\zeta$ of pairs of distinct vertices from $V$, between which a communication must be set up. We are interesting in two special communication patterns:

- **one-to-all**: the connection is from one specified vertex to all others; i.e. $\zeta = \{(r, u) | u \in V, u \neq r\}$, where $r$ is the specified vertex (usually called the source).

- **all-to-all**: the connection is between all pairs of vertices; i.e. $\zeta = \{(u, v) | u, v \in V, u \neq v\}$. 
In the following text, the network is denoted by $G$, and the $\zeta$ is either one-to-all or all-to-all pattern.

**Definition 1** A virtual path layout (shortly VPL) $\Psi$ on $G$ is a collection of simple paths in $G$, called virtual paths (shortly VPs).

From now on, we distinguish between two types of VPLs, depending on their communication pattern $\zeta$, namely one-to-all or all-to-all VPL.

**Definition 2** The load $L(e)$ of an edge $e \in E$ in a VPL $\Psi$ is the number of VPs $\psi \in \Psi$ that include $e$. $L(e)$ is also referred as edge congestion of $e$.

**Definition 3** The maximal edge load $L_{\text{max}}(\Psi)$ of a VPL $\Psi$ is $\max_{e \in E} L(e)$.

**Definition 4** The hop count $H(u,v)$ between two vertices $u, v \in V$ in a VPL $\Psi$ is the minimum number of VPs whose concatenation forms a path in $G$ connecting $u$ and $v$. This concatenation is also called virtual channel (shortly VC). If no such VPs exist, we define $H(u,v) = \infty$.

**Definition 5** The maximal hop count of a VPL $\Psi$ is computed as

$$H_{\text{max}}(\Psi) := \max_{(u,v) \in E} \{H(u,v)\}.$$  

The problem is to design a VPL such that hop count and load are minimized simultaneously. The hop–load tradeoff is studied for various topologies.

We conclude this section by parameters of basic layouts for complete binary trees, known from literature. These shortest path layouts are useful for the design of VPLs on butterfly networks.

**Layout 1**

$H_{\text{max}} = 1$, $L_{\text{max}} = \lfloor N/2 \rfloor$, where $N$ is the number of vertices in a complete binary tree.

**Layout 2**

$H_{\text{max}} = H$, $L = O(N^{1/2})$.

3 VPLs for butterfly network

The design of VPLs for butterfly networks has not yet been studied. However, some results can be obtained as a consequence of known general theorems. We will present some new lower and upper bounds together with related VPL designs.
3.1 Butterfly topology

First, we describe the butterfly topology and introduce our labeling conventions to simplify the rest of the section.

The butterfly network $BF_n$ consists of $n^2 - 1$ vertices, usually represented by $n$ rows and $2n - 1$ columns. Let $v$ be a vertex in the $r$th row and in the $c$th column. Then we label $v$ as $v_{r,c}$. We now define the interconnection of vertices recursively.

$BF_1$ network is a single vertex. Let $C, D$ be two $BF_n$ networks. Construct $BF_n$ as follows. Add 1 to each row of $C$ and $D$, so they now have rows $2n - 1$. Also add $2n - 1$ to each column of $D$. Now add new $2n$ vertices $v_1, \ldots, v_{1,2n}$. Finally, add the following edges to the resulting graph:

- $\forall i, 1 \leq i \leq 2n$, add an edge $(v_{1,i}, v_{2,i})$
- $\forall i, 1 \leq i \leq 2^n - 1$, add an edge $(v_{1,i}, v_{2,i+2^n-1})$
- $\forall i, 2^n - 1 + 1 \leq i \leq 2^n$, add an edge $(v_{1,i}, v_{2,i-2^n+1})$

We will sometimes refer to the rows as levels. The number of rows (levels) is denoted by $n$, and the total number of vertices as $N$, so $N = n^2 - 1$.

**Definition 6** Let $BF_n$ be a butterfly. For any $1 \leq i \leq 2^{n-1}$, define a tree $T_i$ corresponding to the vertex $v_{1,i}$ as follows: Set the vertex $v_{1,i}$ to be the root of the $T_i$; for every vertex $v_{a,b}$ already in $T_i$, recursively add its two neighbours in $BF_n$ from the level $b + 1$, unless $b = n$, where $v_{a,b}$ is a leaf in $T_i$.

Similarly, we can define the tree $T'_i$ corresponding to the vertex $v_{n,i}$ oriented bottom-up. All these trees are complete binary trees with depth $n$ and $2^n - 1 = 2^{N/n} - 1$ vertices. These trees make a backbone used for our VPLs.

3.2 Known results

**Theorem 1** Let $G$ be a graph of order $N$ with $\Delta = O(1)$ and $diam(G) = O(\log N)$. Then for not necessarily shortest path gossip on $G$ it holds $H = O(\frac{\log N}{\log L})$ for any given $L$.

**Proof.** See [5].

The lower bound for this result holds even for one-to-all layouts, so it is tight also for broadcast virtual path layouts. This result can be directly applied to butterfly networks. In this paper, however, we will study the reverse problem for butterfly topology, namely to determine $L$ for given $H$. Moreover, it will turn out, that this reverse relation for one-to-all layouts (i.e. to compute $L$ for given $H$) is more difficult to prove, since it will imply the above result and not vice versa.
3.3 Lower bound

**Theorem 2** Let BF\(_n\) be a butterfly with \(N\) vertices. For every one-to-all VPL with given hop count \(H\) it holds

\[
L / BP \geq \Omega(N^{1/H}).
\]

The proof is omitted and will be included into the full version of the paper. Note that we are not aware of better lower bound for all-to-all case.

3.4 One-to-all VPLs for butterfly networks

In this section we design one-to-all VPLs for butterfly networks. Firstly, we look at one-to-all layouts from the “corner” vertex \(v_{1,1}\). Later on, we generalize these results for any source \(v_{i,j}\).

We start with an asymptotically optimal one-to-all layout from the vertex \(v_{1,1}\) on the butterfly.

**Layout 3**

Let \(BF_n\) be a butterfly network. Let \(H / AL \geq 2\) be an upper bound on the hop count. We construct an one-to-all VPL from \(v_{1,1}\) as follows:

1. Start with empty VPL \(\Psi\).
2. Construct a VPL for complete binary tree \(T_1\) from the vertex \(v_{1,1}\) with upper bound for hop count equal to \(H\) using slightly modified Layout 2. In the Layout 2 we divided the tree \(T_1\) into \(H\) levels of trees with depth equal to \[\left\lfloor \frac{l-1}{2} \right\rfloor + 1\] or \[\left\lfloor \frac{l-1}{2} \right\rfloor + 2\] (\(l\) is the depth of \(T_1\)). In our modification, we increase the depth of trees from Layout 2 by \(lglgN\) (\(N\) is the number of vertices in the whole butterfly) except for the lowermost level. In the lowermost level the depth of trees is decreased by \((H-1)lglgN\), so the number of levels does not change. More precisely, the upper \(H-1\) levels of trees have the depth \[\left\lfloor \frac{l-1}{2} + lglgN \right\rfloor + 1\] and the lowermost level of trees have the depth \[\left\lfloor \frac{l-1}{2} - (H-1)lglgN \right\rfloor + 1\]. See Fig. 1. Finally, add all VPs from the VPL for \(T_1\) to \(\Psi\).
3. For each column \(c\) of \(BF_n\), by \(v_{i,c}\) denote a vertex from \(T_1\) on the column \(c\) with the minimum possible row, i.e. \(i_c = min\{i|v_{i,c} \in T_1\}\). If there exist connections from \(v_{1,1}\) to \(v_{i,c}\) with at most \(H-1\) hops (in the layout for \(T_1\) from step 2), add paths \((v_{i,c}, v_{i-1,c}), \ldots, (v_{i,c}, v_{1,c}, \ldots, v_{1,c})\) to \(\Psi\) (these are one-hop layouts for chains \((v_{i,c}, \ldots, v_{1,c})\)). This is the case, where \(v_{i,c}\) is not in the lowest level of trees in Layout 2 (\(i_c < n-k\)).
(4) For the rest of vertices $v_{i,c}$ (not included in the previous step 3) find the pivot $v_{i,j}$ of their subtree in the Layout 2. Now, for each path $(v_{i,j}, \ldots, v_{i,c})$ already included in $\Psi$ (in fact, there is only one such path), add paths $(v_{i,j}, \ldots, v_{i,c}, v_{i,c-1}, \ldots, v_{1,c})$ to $\Psi$, so there is one-hop connection from the pivot $v_{i,j}$ to any vertex $v_{m,c}$ for $m \leq i_c$.

Analysis of VPL. We need the following lemma:

Lemma 1 Let $BF_n$ be a butterfly. Now take any vertex $v_{i,c} \in BF_n \setminus T_1$. Then $i < i_c$, where $i_c$ is the same as defined in the previous layout.

Proof. Suppose that $i \geq i_c$. From the definition of $i_c$, we get $v_{i,c} \in T_1$. Now, from the definition of $T_1$, all neighbours of $v_{i,c}$ in level $i_c + 1$ are in $T_1$. So $v_{i,c+1} \in T_1$. By induction, any vertex $v_{k,c}$, $k \geq i_c$, is in $T_1$. This is a contradiction since $v_{i,c} \in BF_n \setminus T_1$. 

Now, we show that $H$ hops are sufficient to get from $v_{1,1}$ to any vertex $v_{i,c}$. Consider three cases:

- $v_{i,c} \in T_1$: The fact follows directly from the correctness of Layout 2 and from the fact that it is used in VPL $\Psi$ for the tree $T_1$.

- $v_{i,c} \in BF_n \setminus T_1$ and the corresponding $v_{i,c}$ is reachable from $v_{1,1}$ in at most $H - 1$ hops (using layout for $T_1$): We use the VC from $T_1$ which connects $v_{1,1}$ and $v_{i,c}$, prolonging it with the path $(v_{i,c}, v_{i,c-1}, \ldots, v_{1,c})$ from step 3 of construction. Following Lemma 1 it holds $i < i_c$. 

Figure 1: Tree $T_1$ partitioned into the levels of trees
v_{i,c} \in BF_n \setminus T_1 \) and the corresponding \( v_{i,c} \) is reachable from \( v_{1,1} \) in \( \mathcal{H} \) hops
(using layout for \( T_1 \)): We take the VC from \( T_1 \) which connects \( v_{1,1} \) and \( v_{i,c} \). The last path in this VC is the path \( (v_{m,j}, \ldots, v_{i,c}) \), where \( v_{m,j} \) is the pivot for the subtree (in Layout 2 for \( T_1 \)) which contains the vertex \( v_{i,c} \). From
the step 4 of construction, there is a path \( (v_{m,j}, \ldots, v_{i,c}, v_{i-1,c}, \ldots, v_{i,c}) \) in VPL \( \Psi \). We use this path to replace the previous path \( (v_{m,j}, \ldots, v_{i,c}) \) in VC between
\( v_{1,1} \) and \( v_{i,c} \). The resulting VC connects \( v_{1,1} \) and \( v_{i,c} \) with at most \( H \) hops.

It is easy to check that all above connections are shortest paths in \( BF_n \). Now we
will concentrate on the parameter load.

Paths from step 3 are edge disjoint with any path from other steps. Their worst-case load is \( O(lgN) \).

Now look at the edges in the tree \( T_1 \). For any subtree from the modified Layout 2 for \( T_1 \), except trees on the lowest level, there are only paths from step 2. Their load can be computed as follows:

\[
\text{upper levels: } \mathcal{L} = O(2^{\left(\frac{\lg 2}{\lg N} + \frac{\lg N}{\frac{1}{H}}\right)}) = O(2^{\left(\frac{\lg 2}{\lg N} + \frac{\lg N}{\frac{1}{H}}\right)N}) = O\left(\frac{N}{lgN}\right)^{\frac{1}{H}} lg^\frac{1}{H} N)
\]

since the load of each subtree is equal to the number of its vertices (on upper
levels), that is \( 2^{\left(\frac{\lg 2}{\lg N} + \frac{\lg N}{\frac{1}{H}}\right)N} \), and it holds \( 2^l = \Theta\left(\frac{N}{lgN}\right) \).

For subtrees on the lowest level, each path in step 4 is replaced by at most \( O(lgN) \) paths, so the load is at most:

\[
\mathcal{L} = O(lgN, 2^{\left(\frac{\lg 2}{\lg N} - \frac{\lg N}{\frac{1}{H}}\right)N}) = O(lgN, \frac{2^\frac{\lg N}{\frac{1}{H}}}{2^{\frac{\lg N}{\frac{1}{H}} - \frac{\lg N}{\frac{1}{H}}}}) = O\left(\frac{N}{lgN}\right)^{\frac{1}{H}} lg^\frac{1}{H} N)
\]

Since both these loads are equal, the final load is (the maximum of previous three loads)

\[
\mathcal{L} = O\left(\frac{N}{lgN}\right)^{\frac{1}{H}} lg^\frac{1}{H} N) = O(N^{\frac{1}{H}}),
\]

which is asymptotically optimal layout recalling the lower bound from Theorem 2.

\textbf{Claim 1} Let \( BF_n \) be a butterfly network. Let \( \mathcal{H} \geq 2 \) be an upper bound on the hop count. Due to the previous layout and the Theorem 2, we can construct an one-to-all VPL from \( v_{1,1} \) for \( BF_n \) with \( \mathcal{L} = \Theta(N^{\frac{1}{H}}) \).
Note that the same scheme can be used to broadcast from any vertex of butterfly lying on the first (topmost) level and from any vertex on the last (bottommost) level.

Now, we are ready to present an asymptotically optimal one-to-all VPL from any vertex of the butterfly network. We start with some definitions.

**Definition 7** Let BF\(n\) be a butterfly and \(v_{r,c}\) be any vertex from BF\(n\). Define \(T_{r,c}\) to be a complete binary tree rooted at \(v_{r,c}\) spreading downwards in BF\(n\) (to rows \(r+1, r+2, \ldots, n\)) as in Definition 6. Similarly, define \(T_{r,c}'\) to be a complete binary tree rooted at \(v_{r,c}\) going upwards.

Recalling Definition 6 we have \(T_{1,c} \equiv T_c\) and \(T_{n,c}' \equiv T_c'\).

**Definition 8** Let BF\(n\) be a butterfly and \(v_{r,c}\) be any vertex from BF\(n\). Define \(T_{r,c}[p]\) to be a complete binary tree rooted at \(v_{r,c}\) spreading down to \(p\) levels in BF\(n\). Similarly define \(T_{r,c}'[p]\) which spreads upwards.

The following layout is rather complicated. The reader is suggested to assume that \(K \leq \mathcal{H}\) (in step 2) for the first time and ignore all parts (in the design and analysis) which concern the \(K > \mathcal{H}\) possibility. Once the layout is understood in this way, the \(K > \mathcal{H}\) possibility should be taken into account.

**Layout 4**

Let BF\(n\) be a butterfly network. Let \(\mathcal{H} \geq 2\) be an upper bound on the hop count. Let \(R \in BF_n\) be any vertex of the leftmost column in BF\(n\) (\(R \equiv v_{k,1}\) for some \(1 \leq k \leq n\)). We construct an one-to-all VPL from \(R\) as follows:

1. Start with empty VPL \(\Psi\).
2. Construct a one-to-all VPL for BF\(n\) from the vertex \(v_{n,1}\) with upper bound for hop count \(\mathcal{H}\) using Layout 3. Add all VPs from this VPL to \(\Psi\). This layout divides the butterfly into \(\mathcal{H}\) levels (see Fig. 2). We define \(j\) as \(j = \max\{v_{j,1} \text{ is pivot in Layout 3} \mid j \leq k\}\). Let \(K\) be the number of levels (from Layout 3) below \(v_{j,1}\) in BF\(n\).
   
   \{In our example (Fig. 2), \(K = 4\), since there are three levels of pivots below \(v_{j,1}\). It is possible to have \(K > \mathcal{H}\), in this case \(j = 1\).\}

3. Remove VPs added in step 2 which include vertices from \(\{v_{r,c} \mid k \leq r \leq n \land 1 \leq c \leq 2^{n-k}\}\).
   
   \{These paths are not needed in our construction. However, this step is optional, since leaving these paths in our VPL \(\Psi\) will not affect asymptotical optimality.\}
(4) Construct a VPL for a complete binary tree $T_{k,1}$ (see Definition 7) with hop count $K$ (from step 2) using slightly modified Layout 2. In our modification, we divide the tree $T_{k,1}$ into $K - 1$ levels of height $\frac{n-1}{2^k} + \log_2 N$ (equal to the height of levels in layout for $BF_n$ from step 2) and the last $K$th level with height $\frac{n-1}{2^k} + \log_2 N - (k - j)$ if $K < \mathcal{H}$ or with height $\frac{n-1}{2^k} - (\mathcal{H} - 1) \log_2 N - (K - j)$ if $K = \mathcal{H}$. The $k$ is taken from initial assumptions, $j$ is from step 2. Add these VPs to $\Psi$.

(5) For each column $1 \leq c \leq 2n-k$ of $BF_n$, let $v_{c,e}$ be a vertex from $T_{k,1}$ on column $e$ with the minimum possible row, i.e. $i_c = \min\{i | v_{c,e} \in T_{k,1}\}$. Add paths $(v_{c,e}, v_{c-1,e}, \ldots, v_{k,e}, v_{k-1,e}, \ldots, v_{k,e})$ to $\Psi$ (these are one-hop layouts for chains $(v_{c,e}, \ldots, v_{k,e})$).

(6) For each column $1 \leq c \leq 2n-k$ of $BF_n$ such that there exist connection from $v_{k,1}$ to $v_{c,e}$ with at most $K - 1$ hops (in the layout for $T_{k,1}$), find the pivot $v_{i,y}$ of $v_{c,e}$'s subtree in Layout 2. Add path $(v_{i,y}, v_{i,c}, v_{i-1,c}, \ldots, v_{i,c})$ to $\Psi$. Construct one-hop one-to-all VPL for tree $T'_{k,c}[k - j]$ using Layout 1. Add paths from this VPL to $\Psi$. Moreover, if $K = \mathcal{H}$, for each column $d$ (except of column $c$) of the tree $T'_{k,c}[k - j]$ find a vertex $v_{i,d} \in T'_{k,c}[k - j]$ with maximal possible row $i_d$, i.e. $i_d = \max\{a | v_{a,d} \in T'_{k,c}[k - j]\}$ and add paths

$$(v_{k,c}, \ldots, v_{d,d}, v_{d+1,d}, \ldots, v_{n,d})$$

and $$(v_{k,c}, \ldots, v_{d,d}, v_{d+1,d}, \ldots, v_{n,d})$$
to $\Psi$.

(7) For each $v_{i,c}$, $1 \leq c \leq 2n-k$ such that we can get from $v_{k,1}$ to the $v_{i,c}$ by exactly $K$ hops ($v_{i,c}$ is at the lowest level in Layout 2) find the pivot $v_{i,y}$ of $v_{i,c}$'s subtree in Layout 2. Now

$$\forall v_{a,b} \in T'_{k,c}[k - j] \text{ add path } (v_{i,y}, \ldots, v_{i,c}, v_{i-1,c}, \ldots, v_{k,c}, \ldots, v_{a,b})$$

to $\Psi$. Moreover, if $K = \mathcal{H}$,

$$\forall v_{a,e} \in BF_n, \quad k < a < i_c \text{ add path } (v_{i,y}, \ldots, v_{i,c}, v_{i-1,c}, \ldots, v_{a,e})$$

to $\Psi$, so in this case the chain (see step 5) is reachable directly from $v_{i,y}$. Finally, still only if $K = \mathcal{H}$, for each column $d$ (except of column $c$) of the tree $T'_{k,c}[k - j]$ find a vertex $v_{i,d} \in T'_{k,c}[k - j]$ with maximal possible row $i_d$, i.e.

$$i_d = \max\{a | v_{a,d} \in T'_{k,c}[k - j]\}$$

and add paths

$$(v_{i,y}, \ldots, v_{i,c}, v_{i-1,c}, \ldots, v_{k,c}, \ldots, v_{d,d}, v_{d+1,d}, \ldots)$$
to $\Psi$.

The layout is schematicaly shown in Fig. 2.

**Analysis of VPL.**

**PART A - correctness of VPL**

Let $v_{r,c}$ be any vertex of $BF_n$. It is included in at least one of the following sets:

- **Complete binary tree $T_{k,1}$**: In this case, we can get from the $v_{k,1}$ to the $v_{r,c}$ in at most $K$ hops using layout from step 4. The VPL for complete binary trees uses the shortest paths.

- **Chain ($v_{k,c}, \ldots, v_{k,c}$)**: We can get from $v_{k,1}$ to $v_{k,c}$ in at most $K$ hops (step 4). If $K < \mathcal{H}$ one more hop is needed from $v_{k,c}$ to the $v_{r,c}$ from step 5. The number of hops used is at most $K + 1 \leq \mathcal{H}$. If $K = \mathcal{H}$ find $v_{k,c}$’s pivot $v_{i,y}$ in $T_{k,1}$. We can get from $v_{k,1}$ to $v_{i,y}$ in at most $K - 1$ hops. Since $K = \mathcal{H}$, there is a single hop path from $v_{i,y}$ to $v_{r,c}$ (from step 7). So $\mathcal{H}$ hops are needed in this case.

All these vertices are in “subbutterfly”

$$\{v_{a,b} \mid k \leq a \leq n \land 1 \leq b \leq 2^{n-k}\}$$
and it is easy to verify, that they use shortest path from \(v_{k,1}\) to \(v_{r,c}\).

- **One of the subtrees \(T'_k, c[k - j]\):** We firstly find pivot \(v_{i,j}\) for vertex \(v_{k,b}\) from step 6 or step 7 (this pivot could be also the vertex \(v_{k,1}\) itself). We begin with VPs from \(v_{k,1}\) to \(v_{i,j}\). If it takes \(K - 1\) hops (\(K\) hops to \(v_{k,b}\)), then we can get from \(v_{i,j}\) to \(v_{r,c}\) through vertices \(v_{q,b}\) and \(v_{k,b}\) in one hop (step 7). If, on the other hand, VP from \(v_{k,1}\) to \(v_{i,j}\) takes at most \(K - 2\) hops, we can add two VPs, \(v_{i,j}\) to \(v_{k,b}\) and \(v_{k,b}\) to \(v_{r,c}\) from step 6. In both cases we use at most \(K\) hops. It is important that we can get to any vertex \(v_{j,c}\), \(c \leq 2^{n-j}\) in at most \(K\) hops. It is still easy to see, that we use shortest paths (for complete characterization of the shortest paths in butterfly networks see e.g. [8]).

- **Vertices in \(\{v_{a,b}\}, j \leq a \leq n \land 2^{n-k} < b \leq 2^{n-j}\), not included in the previous step:** We set \(i_r = \max\{a | v_{a,c} \in T'_k, c mod 2^{n-i}[k - j]\}\). Such index exists, since \(v_{j,c} \in T'_k, c mod 2^{n-i}[k - j]\). Now, if \(K < H\), we construct an VP from \(v_{k,1}\) to \(v_{i,s}\) with at most \(K\) hops (see previous step) and add one hop from \(v_{i,s}\) to \(v_{r,c}\) from step 2 of construction for total of \(K + 1 \leq H\) hops. If \(K = H\), let \(b = c\) mod \(2^{n-k}\), so \(v_{i,s} \in T'_k, b[k - j]\). Let \(i_b = \min\{a | v_{a,b} \in T_{k,1}\}\) and let \(v_{i,b}\) be \(v_{i,b}\)'s pivot in tree \(T_{k,1}\). Then there exist a path from \(v_{k,1}\) to \(v_{i,b}\) in at most \(K - 1\) hops (step 4) and a single-hop path from \(v_{i,b}\) to \(v_{r,c}\) through vertices \(v_{k,b}\) and \(v_{r,c}\) from step 7.

- **The rest of vertices:** If \(K = H\) this set is empty, so we can assume, that \(K < H\). These vertices are from the larger part of Fig. 2. Let \(q\) denote the shortest path between \(v_{n,1}\) and \(v_{r,s}\) used in layout from step 2 to connect these vertices. We set \(x = \min\{a | 1 \leq a \leq 2^{n-j} \land v_{j,a} \in q\}\). The minimum operator is only for syntax, since there is exactly one vertex in the specified set. The set is not empty, because the set \(\{v_{j,a} | 1 \leq a \leq 2^{n-j}\}\) is a vertex cut in \(BF_n\), so any path from \(v_{n,1}\) to \(v_{r,s}\) go through it. Firstly we connect \(v_{k,1}\) with \(v_{j,x}\) with at most \(K\) hops (see 3rd item on this list) and from \(v_{j,x}\) to \(v_{r,c}\) we use VPs from step 2 (the rest of the path \(q\)). We can do it, since \(v_{j,x}\) is pivot in layout from step 2. The connection of \(v_{n,1}\) and \(v_{r,c}\) used at most \(H\) hops. The connection from \(v_{n,1}\) to \(v_{j,x}\) uses \(K\) hops, so the rest of the path from \(v_{j,x}\) to \(v_{r,s}\) is in at most \(H - K\) hops. Combining with path from \(v_{k,1}\) to \(v_{j,x}\), we can get from \(v_{k,1}\) to \(v_{r,s}\) in at most \(K + H - K = H\) hops.

PART B - shortest path analysis

**Lemma 2** Let \(v_{r,c} \in \{v_{a,b} | 1 \leq a \leq k \land 1 \leq b \leq 2^{n-1}\}\). Then we can get from \(v_{k,1}\) to \(v_{r,c}\) in at most \(H\) hops using shortest path.

**Proof.** For detailed description of shortest paths in butterfly topology see [8]. Let \(p\) be the shortest path between \(v_{k,1}\) and \(v_{r,c}\). \(v_{r,c} \in \{v_{a,b} | 1 \leq a \leq k \land 1 \leq b \leq 2^{n-1}\}\). There are three possibilities
• The path $p$ does not change direction (each row between $k$ and $r$ is visited exactly once). Combine VPL from step $6$ and $2$ to get $H - K + 1$ hop layout for tree $T'_{k,1}$. Since $v_{rc} \in T'_{k,1}$ (because the path does not change direction), we can use this combined VPL to get from $v_{k,1}$ to $v_{r,c}$. Since it is common VPL for tree, the used path is the shortest one.

• The path $p$ changes (top-down) direction once.
  
  – The path $p$ starts going up (decrease row). It can be transformed into the path $p_2$, which changes column only before changing direction. This can be done due to $r \leq k$. The combined VPL from step $6$ and step $2$ is again useful. If $K < H$, we get firstly from $v_{k,1}$ to $v_{1,c}$ using VPL for $T'_{k,1}$. The rest of the path $p_2$ is straight chain on column $c$. One hop path from the layout in step $2$ can be used to get from $v_{1,c}$ to $v_{r,c}$. If $K = H$ we use only layout from step $6$ and the whole procedure ($v_{k,1}$ to $v_{1,c}$ to $v_{r,c}$) can be done in a single hop.
  
  – The path starts downward (increasing row). This is identical with downward path in the following case.

• The path $p$ changes (top-down) direction twice.
  
  – The path $p$ starts going up (decrease row). From the properties of shortest paths, the path must finish at row $k$ (or below, when $r > k$, but this is not the case of Lemma). Such path can be transformed into path $p_2$, which starts going downwards (the necessary column changes on rows $\geq k$ are taken first). Use $p_2$ in the following case.
  
  – The path $p$ starts going down (increase row). If $v_{r,c} \in \{ v_{a,b} \mid j \leq a \leq k \wedge 1 \leq b \leq 2^{a-j} \}$, we can use connection from part A, the last but one case (the path from that construction has the same length as $p$). Otherwise, we use connection from part A, the last case. Again, the segments between $v_{k,1}$ to $v_{j,c}$ and $v_{j,c}$ to $v_{r,c}$ in the path $p$ might be replaced by equaly long segments from this connection (The rows are not changed, only columns are shifting differently).

This property is exploited in Layout $5$ to get a VPL which uses only the shortest paths for routing.

PART C - load analysis

We will look at the load contributed from each step of construction.

• Step 1. $\mathcal{L} = 0$.

• Step 2. $\mathcal{L} = O(\frac{N}{1gM}(1g\frac{1}{2}N))$ since it is the load of Layout $3$.

• Step 3. $\mathcal{L} = 0$. We only remove paths.
• Step 4. \( \mathcal{L} = O\left( \left\lceil \frac{N}{\lg N} \right\rceil^\frac{1}{3} \lg \frac{1}{N} \right) \) since the largest level of the tree \( T_{k,1} \) has 
\( \frac{n-1}{3} + \log \frac{1}{N} \) rows (see Layout 3).

• Step 5. \( \mathcal{L} = O(\lg N) \), it is the length of chains.

• Step 6. \( \mathcal{L} = O\left( \left\lceil \frac{N}{\lg N} \right\rceil^\frac{1}{3} \lg \frac{1}{N} \right) \). If \( K < \mathcal{H} \), the trees \( T'_{k,c}[k-j] \) have at most 
\( \frac{n-1}{3} + \log \frac{1}{N} \) rows, so \( \mathcal{L} \) is as stated. If \( K = \mathcal{H} \), the trees \( T'_{k,c}[k-j] \) have at most 
\( \frac{n-1}{3} - \log \frac{2^{n-1}}{N} \) rows. To each path at most \( \lg N \) new paths are added, 

hence \( \mathcal{L} = O(\lg N \cdot 2^\left( \frac{n-1}{3} - \log \frac{1}{N} \right) ) \) as stated (see Layout 3).

• Step 7. \( \mathcal{L} = O\left( \left\lceil \frac{N}{\lg N} \right\rceil^\frac{1}{3} \lg \frac{1}{N} \right) \). Let \( a = k - j \) and let \( b \) be the number of rows 
on the lowest level in tree \( T_{k,1} \) from the Layout 2 in step 4 of construction. 

So \( a + b = \frac{n-1}{3} + \log \frac{1}{N} \) if \( K < \mathcal{H} \) and \( a + b = \frac{n-1}{3} - \log \frac{2^{n-1}}{N} \) if \( K = \mathcal{H} \). 

If \( K < \mathcal{H} \), each path from \( T_{i,j}[b] \) is prolonged by at most \( 2^a \) paths (vertices 
of \( T'_{k,c}[a] \)), leading finally to \( \mathcal{L} = O\left( 2^a 2^b \right) = O(2^{a+b}) = O(2^\left( \frac{n-1}{3} + \log \frac{2^N}{N} \right) ) \) 
as stated (see Layout 3). If \( K = \mathcal{H} \), each path from \( T_{i,j}[b] \) is prolonged 
by at most \( 2^a \) paths to \( T'_{k,c}[a] \), which are further prolonged by another 
\( O(\lg N) \) paths to chain \( (v_{id}, \ldots, v_{n,d}) \). Independently, each path from \( T_{i,j}[b] \) 
is prolonged by \( O(\lg N) \) paths to chain \( (v_{id}, \ldots, v_{k,c}) \). So we have fi-
nally \( \mathcal{L} = O(2^a(2^b \lg N + \lg N)) = O(2^{a+b} \lg N) = O(2^\left( \frac{n-1}{3} - \log \frac{2^N}{N} \right) \lg N) \) 
as stated (see Layout 3).

Each step has a load of at most \( O\left( \left\lceil \frac{N}{\lg N} \right\rceil^\frac{1}{3} \lg \frac{1}{N} \right) \), so we have 

\[ \mathcal{L} = O\left( \left\lceil \frac{N}{\lg N} \right\rceil^\frac{1}{3} \lg \frac{1}{N} \right) = O(N^{\frac{1}{3}}). \]

\[ \square \]

**Claim 2** Let \( BF_n \) be a butterfly network. Let \( \mathcal{H} \geq 2 \) be an upper bound on the 
hop count and \( k \) be arbitrary number, \( 1 \leq k \leq n-1 \). Due to the previous layout 
and Theorem 2, we can construct an one-to-all VPL from \( v_{k,1} \) for \( BF_n \) with \( \mathcal{L} = \Theta(N^{\frac{1}{3}}) \).

Note that the same scheme can be used for any vertex \( v_{xc} \) from \( BF_n \) using 
automorphism which maps \( v_{xc} \) into the vertex \( v_{x,1} \).

In the following VPL we exploit previous layout to get asymptotically optimal 
one-to-all VPL from any vertex which uses shortest paths for routing.
Let \( BF_n \) be a butterfly network. Let \( \mathcal{H} \geq 2 \) be an upper bound on the hop-count. Let \( R \in BF_n \) be any vertex of the leftmost column in \( BF_n \) (\( R \equiv v_{k,1} \) for some \( 1 \leq k \leq n \)). We construct an one-to-all VPL from \( R \) as follows:

1. Start with empty VPL \( \Psi \).
2. Construct one-to-all VPL for \( BF_n \) from vertex \( v_{k,1} \) with upper bound for hop count equal to \( \mathcal{H} \) using Layout 4. Add all VPs from it to \( \Psi \).
3. Construct one-to-all VPL for \( BF_n \) from vertex \( v_{n-k,1} \) with upper bound for hop count equal to \( \mathcal{H} \) using Layout 4. Change top-down orientation of \( BF_n \) (use a bijection \( v_{r,c} \rightarrow v_{n-r,1+rev(c-1)} \), where \( rev \) is reverse function for binary numbers. Now \( v_{n-k,1} \) match the vertex \( v_{k,1} \) from the previous step. Add all VPs (after change of orientation) to \( \Psi \).

**Analysis of VPL.** The load \( \mathcal{L} \) is at most twice the load from Layout 4, so it still holds

\[
\mathcal{L} = O\left( \left\lfloor \frac{N}{IgN} \right\rfloor \frac{1}{lg \frac{1}{\mathcal{H}}} N \right) = O(N^{\frac{1}{\mathcal{H}}}).
\]

Similarly, we can still get from \( v_{k,1} \) to any vertex in at most \( \mathcal{H} \) hops, we only have more alternatives.

According to Lemma 2 and layout from step 2, we can get to any vertex \( v_{r,c} \) with \( r \leq k \) using shortest paths. Similarly, according to Lemma 2 and layout from step 3, we can get to any vertex \( v_{r,c} \) with \( r \geq k \) using shortest paths. So we can get to any vertex of \( BF_n \) using the shortest paths.

**Theorem 3** Let \( BF_n \) be a butterfly network. Let \( \mathcal{H} \geq 2 \) be an upper bound on the hop count and \( k \) be arbitrary number \( 1 \leq k \leq n - 1 \). Due to the previous layout and Theorem 2, we can construct an one-to-all VPL from \( v_{k,1} \) for \( BF_n \) with \( \mathcal{L} = \Theta(N^{\frac{1}{\mathcal{H}}}) \) in which the shortest paths are used for routing.

Note that the same scheme can be used for any vertex \( v_{r,c} \) from \( BF_n \) by mapping it firstly into the vertex \( v_{r,1} \).

**References**


**Martin Makúch** is with the Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia. E-mail: makuch@mail.viapvt.sk

**Peter Ružička** is with the Institute of Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia. E-mail: ruzicka@dcs.fmph.uniba.sk
Optimal Scheduling for Disconnected Cooperation*

GRZEGORZ MALEWICZ  
*University of Connecticut, USA

ALEXANDER RUSSELL  
University of Connecticut, USA

ALEX A. SHVARTSMAN  
University of Connecticut and MIT, USA

Abstract

We consider a distributed environment consisting of $n$ processors that need to perform $t$ tasks. We assume that communication is initially unavailable and that processors begin work in isolation. At some unknown point of time an unknown collection of processors may establish communication. Before processors begin communication they execute tasks in the order given by their schedules. Our goal is to schedule work of isolated processors so that when communication is established for the first time, the number of redundantly executed tasks is controlled. We quantify worst case redundancy as a function of processor advancements through their schedules.

In this work we refine and simplify an extant deterministic construction for schedules with $n \leq t$, and we develop a new analysis of its waste. The new analysis shows that for any pair of schedules, the number of redundant tasks can be controlled for the entire range of $t$ tasks. Our new result is asymptotically optimal: the tails of these schedules are within a $1 + O(n^{-\frac{1}{8}})$ factor of the lower bound. We also present two new deterministic constructions one for $t \geq n$, and the other for $t \geq n^{\frac{3}{2}}$, which substantially improve pairwise waste for all prefixes of length $t/\sqrt{n}$, and offer near optimal waste for the tails of the schedules. Finally, we present bounds for waste of any collection of $k \geq 2$ processors for both deterministic and randomized constructions.

Keywords

distributed algorithms, partitionable networks, scheduling, work, design theory

*This research was supported in part by the NSF Grant CCR-9988304 and the AFOSR Grant F49620-99-1-0244. The work of the second author was supported in part by an NSF CAREER Award. The work of the third author was supported in part by an NSF CAREER Award.
1 Introduction

The power and the challenge of distributed computing lie in the ability to marshal decentralized computing resources in seeking a cooperative solution to a common problem. This is frequently difficult because the distributed computing medium may be subject to dynamic changes or failures. In its most basic form, a fundamental problem for distributed computing is to efficiently perform a set of tasks despite the changes in the computing medium. Such a problem has been studied in message-passing models [1, 2, 4], using group communications [3, 6], and in shared-memory models [11]. These studies present various load-balancing techniques for structuring the work for the computing devices that are able to communicate by some means. Other studies investigate the impact of communication topology on the effectiveness of load-balancing [5, 13].

The emergence of mobile computing paradigms has created new dimensions for the problem of performing a collection of tasks in a distributed setting. Indeed, an intrinsic feature of mobile computing [10] is that communication topology changes over time, and some devices may not be able to communicate with others for prolonged periods of time. In many distributed applications computation can progress in only one of the connected components, while other components have to be prevented from acting on their own, for example, to ensure coherence of a shared object. However, there are settings in which any connected component of processors or even an isolated processor may meaningfully carry on with the computation regardless of any other component. For example, this is the case when a set of idempotent tasks (in particular tasks admitting “at least once” execution semantics) needs to be performed in a distributed system. A simple instance of this occurs when a collection of computers is charged with the task of breaking a cryptographic scheme by brute force search of the key space. In such a setting it is interesting to investigate how to best utilize the resources of every processor of the system during the entire computation subject to changing connectivity. These scenarios motivate the following abstract problem.

We consider the abstract problem of performing $t$ independent and idempotent tasks in a distributed environment consisting of $n$ processors where each processor knows all tasks. Initially communication is unavailable, and so each processor must begin to work in isolation from the other processors. At some unknown future time, some unknown subset of the processors may establish communication. When these processors begin to communicate, they can determine the set of tasks $D$ that they have collectively performed prior to this time. Efficiency concerns demand that this set is as large as possible. Hence our goal is to devise algorithms that prescribe to each processor a sequence of tasks to perform when the processors work in isolation, so as to maximize the size of the set $D$ at the instant when the processors establish communication. We pursue the objective of constructing sequences of tasks, called schedules, that have the property that, when processors from different components establish communication, the amount of redundantly
executed tasks, called waste, is small. We want to minimize worst-case waste, i.e.,

waste across all possible collections of processors that may establish commu-
nication, and across all possible instances when communication is established, and

relative speeds of processors. This would enable us to give a deterministic upper

bound on the amount of redundant work.

To illustrate this problem, consider the case of two processors \((n = 2)\). Let the

first processor execute tasks in the order \(\langle 1, 2, 3, \ldots, t \rangle\), and the second pro-
cessor execute them in the reverse order \(\langle t, t-1, t-2, \ldots, 1 \rangle\). In the absence of

communication each processor has no knowledge of what the other is doing. If

the processors are able to communicate after they have completed \(a\) and \(b\) tasks

respectively and if \(a + b \leq t\) then no work is wasted (no task is executed twice).

If \(a + b > t\), then the redundant work is \(a + b - t\). In fact this is a lower bound

on waste for any set of two or more processors. If the two processors have inde-

pendently performed all tasks, then the wasted work is \(t\). Of course this problem

becomes more interesting when we deal with \(n \geq 2\).

This scheduling problem was defined and studied by Dolev et al. [3]. They

showed that if each processor begins its work in isolation, then the waste in-
curred when communication is first established between any two processors can

be bounded by 1 as long as processors have executed no more than \(t^{1/3}\) tasks.

When communication is unavailable for long periods of time, and processors have

finished executing the initial \(t^{1/3}\) tasks from their schedules and now work on the

remaining tasks, their construction does not offer interesting bounds on wasted

work.

Subsequently Malewicz et al. [12] began developing a scheduling theory with

the goal of offering optimal and efficient solutions for this problem. The emerging

time exhibits links between design theory, graph theory, linear algebra, and our

distributed computing problem. We now overview this earlier work [12].

It was shown that set systems called designs can be used to provide each pro-
cessor with a schedule of length about \(\sqrt{t}\). These schedules have pairwise waste

of 1. However such design-theoretic constructions fail to address the scheduling

problem for the complete range of \(t\) tasks. To address the need to schedule

all tasks, it was shown [12] how to schedule the \(t\) tasks such that waste grows

gracefully as processors progress through their schedules. This work extends the

standard \(\Theta(\sqrt{t})\)-length design-theoretic constructions. These long schedules with

controlled waste can be constructed in time linear in the length of the schedule.

The deterministic construction of [12] yields schedules such that pairwise wasted

work increases gradually, but only through the first \(\frac{1}{3}t\) tasks. A lower bound on

pairwise waste was shown [12] to be \(\Omega(ab/t)\) when one processor performs \(a\)

and another \(b\) tasks, i.e., the wasted work must grow quadratically with the advance-
ments of processors. Finally, it was shown that when the schedules are selected

uniformly at random among all permutations on \(\{1, \ldots, t\}\), then such schedules

have asymptotically optimal upper bound on waste.

Overall, the deterministic construction [12] of schedules when the number
of processors $n$ is equal to the number of tasks $t$ has several limitations: (a) for the first $\sqrt{n}$ tasks the pairwise redundant work is linear in the number of tasks, whereas the lower bound for the same range is only 0 or 1(!), (b) the analysis ensures graceful increase in redundant tasks only for the prefixes of about $\frac{1}{2}n$ tasks, and when processors have advanced to element number $a > \frac{4}{3}n$ of their schedules only a trivial bound of $a$ on waste is provided, (c) each processor needs to maintain $\Theta(1)$ memory during construction resulting in the total $\Theta(n^2)$ memory required.

Contributions. This paper focuses on the case when $n \leq t$. We advance the state-of-the-art with the ultimate goal of developing a general theory that yields optimal schedules for task executions in that they minimize redundancy for arbitrary periods of time during which processors work in isolation. The main contributions are as follows:

(i) Efficient construction of optimal schedules. We show in Section 3 how to refine, simplify and extend the deterministic construction of [12] to produce schedules for the case when $n \leq t$. The new analysis shows that for any pair of schedules, the number of redundant tasks can be controlled for the entire range of $t$ tasks. This appears in Theorem 4. While the initial prefix of length $t/\sqrt{n}$ of the schedule may be poorly behaved (i.e., may incur $t/\sqrt{n}$ waste), the tails of the schedules are asymptotically optimal; specifically, they are within a $1 + O(1/\sqrt{n})$ factor of the lower bound of [12]. This is shown in Lemma 2. We present a new incremental construction of the schedules such that each processor can build its schedule in isolation. The total construction costs $O(t)$ time, and essentially every task costs $O(1)$ time to construct. $O(1)$ space is sufficient to perform the construction. This appears in Corollary 1.

(ii) Improved prefix behavior. In Section 4, we present two new deterministic constructions that substantially improve pairwise waste for all prefixes of length $t/\sqrt{n}$, while maintaining near optimal waste for the tails of the schedules. The first construction can be used when $t \geq n$, and it offers tight $t/n$ waste for advancements $t/\sqrt{n}$. This appears in Theorem 5. The second construction, for $t \geq n^{3/2}$, optimizes prefix waste in the region up to $t/\sqrt{n}$, as stated in Lemma 9. Each of these constructions enjoys near optimal waste for the tails of the schedules. The cost of construction includes a substantial initial expense of $O(n^2)$ time, as shown in Lemma 8. This expense can be amortized, however the construction is better suited for off-line applications. Our contributions include a methodology for recursive optimization and analysis of schedule prefix behavior.

(iii) Bounds for collections of arbitrary size. In Section 5 we present bounds on waste collectively incurred by arbitrary subsets of $k$ processors. We give bounds for both deterministic and randomized constructions in Theorems 7 and 8.

All the above constructions work for the case when $n$ is about a power of a prime. The Prime Number Theorem guarantees that primes are sufficiently dense,
and so we can use standard padding techniques without significantly deteriorating the quality of padded schedules.

Note that all our constructions are deterministic. In our setting, there are two reasons to prefer deterministic solutions over randomized. The first, of course, is that they do not require a random coin on the part of the participants. The second is that our deterministic solutions offer worst-case guarantees in comparison to the high-probability guarantees offered by typical randomized constructions.

2 Definition and Models

We consider the abstract setting where $n$ processors need to perform $t$ independent and idempotent tasks. A task is idempotent if the execution of the task yields the same result when it is performed more than once. A task is independent if the result of its execution does not depend on the order in which other tasks are executed. The processors have unique identifiers from the set $[n] = \{1, \ldots, n\}$, and the tasks have unique identifiers from the set $[t] = \{1, \ldots, t\}$. Initially each processor knows the tasks that need to be performed and their identifiers.

2.1 Schedules and Waste

A schedule $L$ is a list $L = \langle \tau_1, \ldots, \tau_c \rangle$ of distinct elements from $[t]$, where $c$ is the length of the schedule $L$. A system of schedules $\mathcal{L}$ is a list of $n$ schedules $\mathcal{L} = \langle L_1, \ldots, L_n \rangle$. When each schedule in the system of schedules $\mathcal{L}$ has the same length $c$, we say that $\mathcal{L}$ is of length $c$. Given a schedule $L$ of length $c$, and $a \geq 0$, we define the prefix schedule $L^a$ to be: $L^a = \langle \tau_1, \ldots, \tau_a \rangle$, if $a < c$, and $L^a = L$, if $a > c$. We define the union (the intersection) of any two schedules to be the corresponding set-theoretic operation on the sets of elements in the schedules. We let $\varnothing$ be the concatenation operator on lists.

To assess the quality of a system of schedules $\mathcal{L}$, we are interested in the worst case redundant work of any collection of processors that establish communication after each processor has performed a certain prefix sequence of tasks from its schedule. In this paper we focus on waste for collections of size 2. We formalize the notion of pairwise waste below. Later (Section 5) we discuss a general case of collections of arbitrary size.

**Definition 1** For a system of schedules $\mathcal{L} = \langle L_1, \ldots, L_n \rangle$ the 2-waste (or waste) $w(\mathcal{L}, a, b)$ for any $a, b \geq 0$ is defined as: $w(\mathcal{L}, a, b) = \max_{i \neq j} |L_i^a \cap L_j^b|$.

2.2 A Design-Theoretic Construction

We describe the construction of a system $\mathcal{L}_n = \langle L_1, \ldots, L_n \rangle$ of $n = q^2 + q + 1$ schedules of length $q + 1$, for any prime power $q = p^k$, with elements from $[n]$. 
The construction is based on the designs induced by the lattice of linear subspaces in $\text{GF}(q)^3$, where $\text{GF}(q)^3$ is treated as a vector space over the finite field $\text{GF}(q)$. (The resulting set system is referred to as a $2-(n,q+1,1)$ design in the design theory literature.) These schedules will be used repeatedly in the sequel.

The schedule for processor $1 \leq u \leq n$ is constructed as follows. There are $q^2 + q + 1$ distinct one dimensional subspaces of $\text{GF}(q)^3$, which we denote $\{\ell_1, \ldots, \ell_n\}$. We say that two subspaces $\ell_i$ and $\ell_j$ are orthogonal if $\forall u \in \ell_1, \forall v \in \ell_2, \langle u, v \rangle = \sum u_jv_j \mod p = 0$. It is a fact that for any one dimensional subspace there are exactly $q+1$ one dimensional subspaces that are orthogonal to this subspace. The schedule $L_u$ is defined as a list of all these (one dimensional) orthogonal subspaces, i.e., $L_u = \{\ell_{u1}, \ldots, \ell_{u(q+1)}\}$ such that any one dimensional subspace $\ell_{ui}$, $i = 1, \ldots, q+1$, is orthogonal to $\ell_u$. It is easy to show that any pair of such schedules intersect at a single point (subspace). See [9] for a proof and more discussion.

We record these facts in the following theorem.

**Theorem 1** If $n = q^2 + q + 1$, where $q$ is a power of a prime, then it is possible to construct a system of schedules $L_n = \langle L_1, \ldots, L_n \rangle$ of length $q+1$, such that $w(L_n, q+1, q+1) = 1$, and any element is in exactly $q+1$ distinct schedules. Each schedule can be constructed in isolation. If $q$ is prime, the first element of each schedule can be calculated in $O(\log n)$ time; each subsequent element can be calculated in $O(1)$ time.

(We assume throughout that addition or multiplication of two log($\max\{n,t\}$)-bit numbers can be performed in $O(1)$ time. We also assume that a number $x \in [\max\{n,t\}]$ takes $O(1)$ space.)

### 3 Efficient Construction of Optimal Schedules

In this section we refine, simplify and extend the deterministic construction of [12] to produce schedules for the case when $n \leq t$. The new analysis shows that for any pair of schedules, the number of redundant tasks can be controlled for the entire range of $t$ tasks. We also discuss a novel feature of this construction: the tails of the schedules are asymptotically optimal. Essentially every task costs $O(1)$ time to build, and $O(1)$ memory is sufficient to perform the construction.

#### 3.1 Schedules for $n = t$

We construct a system of schedules of length $n$ by arranging tasks from several schedules of $L_n$ in a recursive fashion. (Recall that $L_n$ only provides schedules of length roughly $\sqrt{n}$.) In preparation for the recursive construction, we record the following lemma about the pairwise intersections of the schedules in $L_n$ indexed by a schedule $L_u$. 


Lemma 1 Let \( L_n = \langle L_1, \ldots, L_n \rangle \) be the system of schedules constructed in Theorem 1, and let \( L_u = \langle t_u^1, \ldots, t_u^{q+1} \rangle, \) \( 1 \leq u \leq n. \) Then for any \( i \neq j, \) we have \( L_{t_u^i} \cap L_{t_u^j} = \{ u \}. \)

Proof. Let us take any two distinct schedules \( L_{t_u^i} \) and \( L_{t_u^j}, \) \( i \neq j. \) By the construction these schedules contain indices of all one dimensional subspaces that are orthogonal to lines \( t_u^i \) and \( t_u^j \) respectively (as defined in Section 2.2). Since \( t_u^i \) is in schedule \( L_u, \) line \( t_u^i \) is orthogonal to line \( t_u. \) By the same argument line \( t_u^j \) is orthogonal to line \( t_u. \) Hence \( u \) is in the schedule \( L_{t_u^i} \) and also in the schedule \( L_{t_u^j}. \) Moreover the intersection \( L_{t_u^i} \cap L_{t_u^j} \) cannot contain any other element because \( w(L_u, q + 1, q + 1) = 1 \) by Theorem 1.

As a result of this lemma, there is only a single repeated element in \( L_{t_u^1}, \ldots, L_{t_u^{q+1}}; \) this element is \( u. \) We use this fact to construct a system of schedules \( P_n \) as follows: \( P_n = \langle P_1, \ldots, P_n \rangle, \) where each schedule \( P_u, 1 \leq u \leq n, \) is defined as \( P_u = \langle u \rangle \circ (\bigcup_{i \in L_u}(L_u - \{ u \})). \) (The big circle denotes concatenation.) Note that each \( P_u \) is a permutation on \([n]\) as it contains \( 1 + (q + 1)q = n \) distinct elements. We conceptually divide any schedule \( P_u \) into \( q + 1 \) segments of elements. The first segment contains the first \( q + 1 \) elements; the remaining \( q \) segments contain \( q \) consecutive elements each.

Theorem 2 Let \( q \) be a prime power, and \( n = q^2 + q + 1. \) For any \( 1 \leq u \leq n, \) the schedule \( P_u \) is a permutation on \([n].\) The first elements of different schedules of \( P_n \) are distinct.

This construction can be done incrementally. For each schedule the first element can be calculated in \( O(1) \) time. For the remaining \( q(q + 1) \) elements, at the beginning of every sequence of \( q \) elements we need to invert at most two elements in \( GF(q). \) When \( q \) is prime this can be done in \( O(\log n) \) using the extended Euclidean algorithm. Other elements of the schedule can be found in \( O(1) \) time. In order to produce consecutive elements of the schedule it is enough to maintain \( O(1) \) state space. Thus finding all elements of the schedule can be done in \( O(1) \) space. We summarize this observation.

Claim 1 Let \( q \) be prime and \( n = q^2 + q + 1. \) For any \( 1 \leq i \leq n \) the elements of \( P_i \) can be calculated efficiently in isolation: to find element number \( 2 + jq, \) \( j = 0, \ldots, q, \) \( O(\log n) \) time is sufficient; all other elements can be found in \( O(1) \) time. The entire construction takes \( O(1) \) space.

Now we show an upper bound on the 2-waste of the system of schedules \( P_n. \)
Theorem 3 Let \( q \) be a prime power, \( n = q^2 + q + 1 \). Let \( a = 1+iq, b = 1+jq, 0 \leq i, j \leq q+1 \). Then

\[
w(\mathcal{P}_n, a, b) \leq \begin{cases} 
0, & i + j = 0, \\
1, & i = 0, j \geq 1 \text{ or } i \geq 1, j = 0, \\
q + ij, & i \cdot j \geq 1.
\end{cases}
\]

Proof. Consider any two schedules \( L \) and \( L' \) of \( \mathcal{P}_n \). As the first elements in these schedules are distinct, the intersection for \( i = j = 0 \) is zero. Assume that \( i = 0 \). Since a schedule is a permutation, the first element of \( L \) can overlap with only one element from \( L' \). The case when \( j = 0 \) is similar. Assume that \( i, j \geq 1 \). Consider the \( i \cdot j \) pairs of segments \((I, J)\), that \( I \) (or \( J \)) is one of the first \( i \) (or \( j \)) segments of \( L \) (or \( L' \)). The recursive construction guarantees that only one pair may have segments where \( I \subseteq J \) or \( J \subseteq I \). For this pair the overlap is at most \( q+1 \) because these may be the first segments in the schedules. For the remaining \( ij - 1 \) pairs the overlap is at most 1. The result follows. \( \square \)

3.2 Scheduling Many Tasks \((t > n)\)

We now consider the case when the number of tasks exceeds the number of processors, i.e., \( t > n \). If \( t \) is substantially larger than \( n \), then it is easy to provide schedules such that processors can do substantial work, i.e., \( t/n \), without overlaps. However it is challenging to extend such trivial schedules beyond \( t/n \). One approach is to group tasks in chunks of \( t/n \) tasks, and use the machinery from previous section, i.e., the system \( \mathcal{P}_n \), to order the execution of chunks. We call the resulting system of schedules \( \mathcal{V}_t \). We define a notion of segments for these schedules in a fashion similar to that used in the above analysis of \( P_t \); the first segment is chosen to contain \( (q+1)t/n \) tasks, and the subsequent segments contain \( qt/n \) tasks each. Theorem 3 immediately yields a bound on waste for these schedules. In the next theorem we show that for any pair of schedules, the number of redundant tasks can be controlled for the entire range of \( t \) tasks.

Theorem 4 Let \( q \) be a prime power, \( n = q^2 + q + 1 \). Let \( a = t/n(1+iq), b = t/n(1+jq), 0 \leq i, j \leq q+1 \). Then

\[
w(\mathcal{V}_t, a, b) \leq \begin{cases} 
t/n, & i + j = 0, \\
q t/n + ij t/n, & i = 0, j \geq 1 \text{ or } i \geq 1, j = 0, \\
i \cdot j \geq 1.
\end{cases}
\]

In the above theorem the granularity of schedule prefixes is \( qt/n \). To bound waste at higher granularity, let \( a = t/n(1+iq) + k, b = t/n(1+iq) + h, 0 \leq k, h < qt/n \). Of course we can bound waste for \( a \) and \( b \) by the waste for advancements \( \mathbf{a} = t/n(1+iq) + qt/n \), and \( \mathbf{b} = t/n(1+iq) + qt/n \). Let \( \mathbf{a} = t/n(1+iq) \),
\( b = t/n(1 + iq) \). Observe that in the worst case the first \( k \) tasks beyond \( a \) and \( h \) beyond \( b \) are wasted. Hence \( w(V_i', a, b) \leq \min \{ w(V_i', b, b), w(V_i', a, h) \} + k + h \).

A direct consequence of Claim 1 is that the construction of system \( V_i' \) can be done incrementally, when \( q \) is prime it is efficient, and is fully amortized: the total cost is \( O(t + \sqrt{n} \cdot \log n) \), where the additive overhead is independent of \( t \). In the next corollary we show that essentially every task costs \( O(1) \) time and space to construct.

**Corollary 1** Let \( q \) be prime, and \( n = q^2 + q + 1 \). For any \( 1 \leq i \leq n \) the elements of \( V_i \) can be calculated efficiently in isolation: to find elements number \( t/n(1 + jq) + 1 \), \( j = 0, \ldots, q \), \( O(\log n) \) time is sufficient, all other elements can be found in \( O(1) \) time. The entire construction can be done in \( O(1) \) space.

### 3.3 Optimality Analysis

In this section we quantify the quality of the constructions from previous sections by contrasting their waste with the lower bound [12]:

**Proposition 1** ([12]) Let \( D_n \) be any system of \( n \) schedules of length \( c \) with elements from \([t]\), and let \( 0 \leq a \leq c, 0 \leq b \leq c \). Then the lower bound for \( w(D_n, a, b) \) is \( LB(a, b) = \frac{n}{t(n-1)}ab - \frac{\min(a,b)}{n} \).

We now compare \( w(V_i', a, a) \) with \( LB(a, a) \). When \( a \leq t/n \), then by Theorem 4 we obtain \( w(V_i', a, a) = 0 \). Hence we have a tight bound for the region \([0, t/n]\). In the next theorem we show that \( w(V_i', a, a) \) is asymptotically quite close to \( LB(a, a) \) for all advancements \( a \) at least \( tn^{-1/4}(1 + o(1)) \).

**Lemma 2** Let \( n = q^2 + q + 1 \geq n \), and let \( V_i' \) be a system of schedules as constructed in Section 3.2. For any \( a \) such that \( \frac{a}{n}(1 + q^{7/4}) \leq a \leq t \) waste

\[
w(V_i', a, a) \leq \left( 1 + O(n^{-1/4}) \right) LB(a, a).
\]

**Proof.** Take any \( \frac{a}{n}(1 + q^{7/4}) \leq a \leq t \). It can be written as \( a = \frac{t}{n} + iq \frac{t}{n} + k \). By Theorem 4 the waste is bounded by \( UB = w(V_i', a, a) \leq \frac{n}{t(n-1)}(q + (i+1)^2) \). Using Proposition 1 we obtain: \( w(V_i', a, a) \geq LB = LB(t/n(1 + iq), t/n(1 + iq)) = \frac{n}{i(n-1)}ab - \frac{\min(a,b)}{n} \). Let us consider the ratio of the upper bound to the lower bound \( \frac{UB}{LB} \leq \frac{(q+1)(q+2)(q+1)}{n(q+1)} \). Observe that by assumption \( i \geq q^{1/4} \), so \( q \leq \frac{2}{\sqrt{q}} \). Observe also that \( 2i + 1 \leq 4q \). From this we get: \( \frac{UB}{LB} \leq \frac{(q+1)(q^2+4q+4)}{p^2q} \leq \frac{(q+1)(q^{5/2}+4q-2)^2}{4q} \leq q^{5/2}+5q^{-1/2} \). Hence \( UB/LB \leq 1 + 5q^{-1/2} + q^{-1} + 5q^{-1/2} \), and the result follows. \( \square \)
Remark. Another way to construct schedules for \( t \geq n \) is to construct the system \( \Pi_t \) and only take \( n \) schedules, say \( P_1, \ldots, P_n \). We call this system \( \Psi_t \). As we show in Lemma 3, \( \Psi_t \) must incur quadratic waste \( \Omega(ab/t) \). This is a disadvantage when \( t \) is much larger than \( n \) because waste is non zero even for small advancements. We show in Lemma 4 that if \( n \leq t^{1/2-\varepsilon} \), \( 0 < \varepsilon < \frac{1}{2} \), then waste of \( \Psi_t \) is \( \Omega(t^{2\varepsilon}) \) when processors execute \( t^{1/2+\varepsilon} \) tasks. Observe that for this region \( \Psi_t \) offers zero waste. The proofs of these lemmas are omitted from this extended abstract.

Lemma 3 Let \( \Psi_t \) be a system of \( n \) schedules for \( t = \hat{q}^2 + \hat{q} + 1 \) tasks, \( \hat{q} \) prime power, \( 2 \leq n \leq t \) and let \( a = 1 + i\hat{q} \), \( b = 1 + j\hat{q} \). Then waste is at least \( w(\Psi_t, a, b) \geq (i-1)(j-1) \).

Lemma 4 Let \( t = \hat{q}^2 + \hat{q} + 1 \), \( \hat{q} \) prime power, \( 2 \leq n \leq t^{1/2-\varepsilon} \), \( 0 < \varepsilon < \frac{1}{2} \). Then the system \( \Psi_t \) is away from the optimum achieved by \( \Psi_t \) by \( \Theta(t^{2\varepsilon}) \) after having executed \( t^{1/2+\varepsilon} \) tasks.

4 Trading Construction Cost for Schedule Quality

When the number of tasks \( t \) is big compared to \( n \), it is more likely that processors establish communication when they execute only some initial section of their schedules. Therefore it is desired to produce schedules with tightly-controlled waste for the first part of schedules. This idea motivates the results presented in this section. We introduce two new deterministic constructions that substantially improve pairwise waste for all prefixes of length \( t/\sqrt{n} \), while maintaining near optimal waste for the tails of the schedules. These constructions include a substantial initial expense of \( O(n^2) \) time and are better suited for off-line applications. Our contributions include a methodology for recursive optimization of schedules and analysis of schedule prefix behavior.

4.1 Controlling Waste for Short Prefixes

In this section we show the first of the two constructions for \( t \geq n \) that offers tight \( t/n \) waste for advancements \( t/\sqrt{n} \). In previous section we showed that the system \( \Psi_t \) is optimal for prefix schedules of length \( a \leq t/n \), and asymptotically well-behaved when \( a \geq \Omega(tn^{-1/8}) \). We desire schedules that are nearly optimal for a broader range of prefix schedule lengths. One disadvantage of \( \Psi_t \) is that the first segment may repeat, so \( t/n(q+1) \) waste may be incurred when a prefix of length \( a = t/n(q+1) \) is executed. To postpone this increase we will sequence segments so that the first segment is distinct across schedules. Our goal is to find a permutation \( \pi_n = (s_1, \ldots, s_n) \) on \([n]\) such that the schedule \( L_n \) contains element \( s_a \). In other words \( \ell_i \) must be orthogonal to \( \ell_{\pi_i}(j) \). This permutation can then be used to select distinct segments as the first segments of schedules in \( \Psi_t \).
Consider the bipartite graph $G = (U, V, E)$ where $U = V = [n]$ and $n = q^2 + q + 1$; here $q$ is a prime power. The set $U$ represents the schedules of $L_n$ and the set $V$ represents elements from the schedules. An edge is placed between $u \in U$ and $v \in V$ when element $v$ is in schedule $L_u$. Based on the structure of $GF(q)^3$, one can show that $G$ is $(q + 1)$-regular (see Theorem 1). By Hall’s theorem (see, e.g., [7]), there is always a perfect matching in a $d$-regular bipartite graph. Observe that such matching yields a permutation $\pi_n$ with the desired properties. In particular if the edge $(u, v)$ appears in the perfect matching, then we put $\pi_n(u) = v$. We find this matching using the Hopcroft-Karp algorithm [8] that runs in time $O(\sqrt{|U| + |V|})$. This yields the following lemma.

**Lemma 5** For the system of schedules $L_n$, the permutation $\pi_n$ on $[n]$ such that $\pi_n(i) \in L_i$ can be constructed in $O(n^2)$ time.

We use $\pi_n$ to construct the system of $n$ schedules $G_n = \langle G_1, \ldots, G_n \rangle$ for $n$ tasks such that the first segments are distinct. Specifically, given $L_n$, the system of schedules $G_n$ is defined as follows. For any $1 \leq u \leq n$, the schedule $G_u$ is given by $G_u = \{u\} \circ (L_{su} - \{u\}) \circ (\bigcup_{i \in L_u - \{su\}} (L_i - \{u\}))$.

**Theorem 5** Let $q$ be a prime power, $n = q^2 + q + 1$. Let $a = 1 + iq$, $b = 1 + jq$, $0 \leq i, j \leq q + 1$. Then: $w(G_n, a, b) \leq \begin{cases} 0, & i + j = 0, \\ 1, & i = 0, j \geq 1 \text{ or } i \geq 1, j = 0, \\ 1, & i \cdot j = 1, \\ q + ij, & i \cdot j > 1. \end{cases}$

**Proof.** Take any two schedules of $G_n$. When $i = j = 1$ observe that by the construction of $G_n$ the first segments of the schedules are distinct. They comprise tasks from $L_i$ and $L_j$ respectively in some order for some $i \neq j$. By the property of the system $L_n$ their overlap is 1. For the remaining cases the argument is the same as in Theorem 3. \qed

Observe that this construction is time-optimal as it produces $n^2$ elements and runs in $O(n^2)$ time. However our algorithm requires $O(n^2)$ time to construct a single permutation in isolation.

**Lemma 6** Let $q$ be prime, and $n = q^2 + q + 1$. For any $1 \leq i \leq n$ the elements of $G_i$ can be calculated in isolation: the first element can be found in $O(1)$ time; the second element can be found in $O(n^2)$ time. To find element number $2 + jq$, $j = 1, \ldots, q$, $O(\log n)$ time is sufficient. All other elements can be calculated in $O(1)$ time.

We could use the system $G_n$ to produce a system of schedules for $t \geq n$ in the same fashion as we produced $V_t$, i.e., by grouping tasks into $t/n$ size chunks.
However this is not a good idea if we want to control waste for small advancements. We know that first segments of schedules from $G_n$ overlap by 1. Hence among the first $(q + 1) \cdot \frac{t}{q}$ elements of the segment, $\frac{t}{q}$ elements are the same as in some other first segment. In the worst case an undesired linear growth of waste could be incurred for advancements between $t/n$ and $2t/n$. One way to alleviate the problem is to execute tasks from the chunks in a different order. Observe that when at most $d$ elements in a segment of $G_n$ are wasted, then $q - d$ are not wasted. Hence we can execute the first tasks from all the chunks, then the second tasks, and so on, up to $t/n$-th tasks from the chunks. We call the resulting system of schedules $H_t$. Notice that this construction has the same additive overhead as the construction of $G_n$, i.e., the total cost is $O(t + n^2 + \sqrt{n} \cdot \log n)$, where the additive overhead is independent of $t$. Observe that this construction yields a slower growth of waste for the initial $(q + 1)t/n$ prefix:

**Lemma 7** Let $n = q^2 + q + 1$, $q$ prime power, $t \geq n$, and let $0 \leq a \leq (q + 1)t/n$. The waste for the system of schedules $H_t$ is bounded by $\frac{t}{q+1} \leq w(H_t, a, a) \leq \frac{t}{q+1} a^2 - \frac{1}{n^2} a$. Our goal is to construct schedules with improved waste in this region when advancements do not exceed $(q + 1)t/n$.

### 4.2 A Recursive Method for the Initial Segment when $t \geq n^{3/2}$

In this section we show the second of the two constructions that improve schedule prefix behavior. This method reschedules elements in the prefixes of length $t/\sqrt{n}$, and the resulting schedules are asymptotically optimal in this range. The method can be applied recursively to tighten the bound on waste in the first segments. Due to lack of space we discuss only the first recursive step in detail.

When the number of tasks $t$ is sufficiently larger than the number of processors $n$, then $H_t$ does not offer good control over waste for advancements $a = O(t/n^{1/2})$. The waste grows linearly and diverges increasingly from the lower bound as the ratio $t/n$ grows. Therefore it is desired to construct a system of schedules that would provide control over waste when $t/n$ is large. In this section we show a construction of such a system of schedules $M_t = \langle M_1, \ldots, M_n \rangle$ for $t = \Omega(n^{3/2})$ tasks. This system provides controlled waste for large advancements in the same way $H_t$ does. In addition we show that we can control waste for prefixes of length up to $(q + 1)t/n$. This gives an improvement over system $H_t$.

For the construction of $M_t$ we assume that $n = q^2 + q + 1$, $q$ prime power, $t/n \geq q + 1$, $q + 1 = \tilde{q}^2 + \tilde{q} + 1$, $\tilde{q}$ prime power. Let $G_n = \langle G_1, \ldots, G_n \rangle$ be the system of schedules constructed in Theorem 6, and let $\pi_n$ be the permutation used in the construction of $G_n$. A schedule $M_t$ is constructed as follows. Let
$G_i = \langle g_1, \ldots, g_n \rangle$. We construct a matrix $M$ with $t/n$ rows and $q + 1$ columns. Column $k$ is filled as follows. Let $i_1 < \ldots < i_{q+1}$ be all the numbers such that $L_{i_j}, 1 \leq j \leq q + 1$, contains the number $g_k$ (their existence is guaranteed by Theorem 1). Let $j$ be such that $i_j = \pi_n(i)$. Let $\mathcal{V}_{i/n} = \langle V_1, \ldots, V_{q+1} \rangle$ be a system of $q + 1$ schedules for $t/n \geq q + 1$ tasks constructed in Corollary 1, and let $V_j = \langle \tau_1, \ldots, \tau_i/n \rangle$. A location $(a, k)$ in the matrix is filled with number $m_{i,k} = (g_k - 1)t/n + \tau_a$. The first $(q + 1)t/n$ tasks of schedule $M_i = \langle m_1, \ldots, m_t \rangle$ are defined to be $\langle m_{1,1}, \ldots, m_{1,q+1}, m_{2,1}, \ldots, m_{2,q+1}, \ldots, m_{i/n,1}, \ldots, m_{i/n,q+1} \rangle$. The remaining tasks of the schedule $M_i$ are defined as $m_{i(k-1)t/n+a} = (g_k - 1)t/n + a, k = q + 2, \ldots, n, a = 1, \ldots, t/n$. The next lemma analyzes the complexity of the construction (see also Figure 1). Due to lack of space the proof is omitted.

**Lemma 8** Let $q$ be prime, $n = q^2 + q + 1$, $t \geq n(q + 1)$, $q + 1 = \hat{q}^2 + \hat{q} + 1$, $\hat{q}$ prime. For any $1 \leq i \leq n$ the elements of $M_i$ can be calculated in isolation and incrementally. The total construction time is $O(t + n^2)$.

The initial cost of $O(n^2)$ is fully amortized if we can do the construction offline for all schedules (the permutation $\pi_n$ can be calculated once for all schedules). Note that if $t \geq n^2$ then the construction is amortized even if done in isolation.

By the construction of system $\mathcal{M}_t$, Theorem 4 gives us an immediate bound on waste. However we can provide a better bound for $\mathcal{M}_t$ when advancements do not exceed $(q + 1)t/n$.

**Theorem 6** Let $q$ be prime power, $n = q^2 + q + 1$, $t \geq n(q + 1)$, $q + 1 = \hat{q}^2 + \hat{q} + 1$, $\hat{q}$ prime power. Let $a = i(q + 1)$, $b = j(q + 1)$, $0 \leq i, j \leq t/n$. Let $t = \frac{i}{m_{q+1}^2} (1 + \hat{q}i)$, and $j = \frac{j}{m_{q+1}^2} (1 + \hat{q}j)$, $0 \leq i, j \leq \hat{q} + 1$. The the waste for the system of schedules $\mathcal{M}_t$ is bounded by:

$$w(\mathcal{M}_t, a, b) \leq \begin{cases} 0 & i + j = 0, \\ \frac{i}{m_{q+1}^2} & i = 0, j \geq 1 \text{ or } i \geq 1, j = 0, \\ \frac{(\hat{q} + \hat{q}j)}{m_{q+1}^2} & i \cdot j \geq 1. \end{cases}$$

**Proof.** Pick any two schedules $M_i$ and $M_j$, $x \neq y$, and let $M$ and $M'$ be the matrices constructed for the schedules as above. By the construction of $\mathcal{M}_t$ after having executed task $a$ and $b$ of some two schedules from $\mathcal{M}_t$, at most task number $i$ and $j$ respectively of some two columns have been executed. By the construction, for any two prefixes of size $(q + 1)t/n$ only $t/n$ tasks are the same. These tasks are located in a column of a matrix $M$ and a column of matrix $M'$. Note that by the construction of the matrices tasks from each of the two columns were executed according to different schedules from the system $\mathcal{V}_{i/n}$. Hence we can use Theorem 4 to bound the waste incurred.

Observe that the term $\hat{q}i$ results in quadratic growth. So we constructed schedules with quadratic growth of waste for the prefixes of length up to $(q + 1)t/n$. We
Lemma 9. Let $q$ be prime power, $n = q^2 + q + 1$, $t \geq n(q+1)$, $q + 1 = \bar{q}^2 + \bar{q} + 1$, $\bar{q}$ prime power, $q^{2/3} \geq 2$. Let $\mathcal{M}_i$ be a system of schedules as constructed above. For any $a$ such that $\frac{t}{n}(1 + \bar{q}^{2/3}) \leq a \leq \frac{t}{n}(q + 1)$ waste
\[ w(\mathcal{M}_i, a, a) \leq \left(1 + O(n^{-1/2})\right) LB(a, a). \]

Proof. We use the matrix $M$ constructed above. Pick any $a$ satisfying the condition from the statement of the theorem. The element number $a$ belongs to a column of $M$ and some row $i$. This column can be divided into consecutive chunks of size $\frac{1}{n(q+1)}$, and there exists $\bar{i}$ such that $\bar{i} := \frac{t}{n(q+1)} \leq \bar{i} \leq \frac{1}{n(q+1)}$. Let $\bar{i} = \bar{i}(q+1)$, and $a = \bar{a}(q+1)$. By the construction we have $\bar{a} < a \leq \bar{a}$, and by selection of $a$, $0 \leq \bar{i} \leq \bar{q} + 1$. Of course waste for $a$ is at most the value of waste for $\bar{a}$, and we can use Theorem 6 to bound it: $UB = w(\mathcal{M}_i, a, a) \leq w(\mathcal{M}_i, \bar{a}, \bar{a}) \leq (\bar{q} + \bar{q}^{2/3}) \frac{t}{n(q+1)}$. Of course the waste for $a$ cannot be smaller than the value of lower bound for $\bar{a}$, and we can use lower bound Theorem 1 to bound it:
\[ LB \geq \frac{\bar{a}}{\pi q^2} \geq \frac{\bar{a}}{\pi n} \geq \frac{\bar{a}}{n(q+1)} \left(\frac{t}{n(q+1)}\right)^2 - \frac{\bar{a}}{n(q+1)^2} = \frac{\bar{a}}{n(q+1)} \left(\bar{i} - 1\right)^2 \bar{q}^2 + \bar{i} - 1). \]

Consider the
upper bound to lower bound ratio. Using the above bounds we bound the ratio:
\[
\frac{UB}{LB} \leq \frac{(n-1)(q+1)^2}{(q+1)^2} \leq \frac{(n-1)}{(q+1)^2} + \frac{\tilde{q}^2}{(q+1)^2} = \left(1 + \frac{1}{q}\right) + \frac{2}{(q+1)^2} + \frac{\tilde{q}+1}{(q+1)^2}.
\]
By the selection of \(a\), \(\tilde{q}^2 \leq \hat{i}\), and by assumption \(1 \leq \frac{1}{2}\tilde{q}^{3/2}\), so we can bound the ratio
\[
\frac{UB}{LB} < \left(1 + \frac{1}{q}\right)^2 + \frac{2}{(q+1)^2} + \frac{\tilde{q}+1}{(q+1)^2} = 1 + O\left(\frac{1}{q^2}\right).
\]
Thus the result follows. \(\square\)

Observe that Theorem 2 also holds for the system \(\mathcal{M}_i\). Hence \(\mathcal{M}_i\) is asymptotically optimal when \(a \geq tn^{-1/8}(1+o(1))\), and also for small prefixes when \(tn^{-7/12}(1+o(1)) < a < tn^{-1/2}(1-o(1))\).

The first step of recursion described above can be used to have even better control over initial waste by applying this step more times. We used \(\mathcal{M}_{i/n}\) in the construction of \(\mathcal{M}_i\) to control in the tail of the prefix. However we could use \(\mathcal{M}_{i/n}\) instead. This recursive process can be iterated as long as \(t\) is sufficiently large compared to \(n\), and \(n\) is sufficiently large. We can use Theorem 9 to relate waste of the resulting system to the lower bound for small advancements.

5 Bounds for collections of arbitrary size

So far we have focused on controlling waste when any two processors establish communication for the first time. It is also interesting to limit this redundancy for any collection of \(k > 2\) processors that begin communication. This bound is captured by the concept of \(k\)-waste. For given system of schedules \(\mathcal{L}\) and numbers \(a_1, \ldots, a_k\) the \(k\)-waste measures the worst case number of redundant tasks executed by a collection of any \(k\) processors that have advanced up to elements number \(a_1\) thru \(a_k\) respectively of their schedules. We denote this by \(w(\mathcal{L}, a_1, \ldots, a_k)\).

It can be shown that \(k\)-waste for the system \(\mathcal{P}_n\) is bounded. Specifically:

**Theorem 7** For a system of schedules \(\mathcal{P}_n\) and numbers \(a_1 = r + i_1q, \ldots, a_k = r + i_kq\) the \(k\)-waste is bounded by \(w(\mathcal{P}_n, a_1, \ldots, a_k) \leq A \cdot (q+1) + B \cdot (B-1)\), where \(S = k + i_1 + \ldots + i_k\), \(A = \min\{S, \left(\frac{k}{2}\right)\}\), \(B = \max\{0, S - \left(\frac{k}{2}\right)\}\).

Furthermore for randomized schedules, a martingale argument shows that \(k\)-waste is close to the expected \(k\)-waste with high probability:

**Theorem 8** Let \(X_1, \ldots, X_n\) be \(n\) permutations of \([t]\), chosen independently at random in the set of all permutations of \([t]\), and let \(a_1 = \ldots = a_k = a \leq t\). Let \(X = \{X_1, \ldots, X_n\}\) be a system of schedules. Then with probability at least \(1 - 1/n\), the \(k\)-waste of the system \(X\) for numbers \(a_1, \ldots, a_k\) is at most \(w(X, a_1, \ldots, a_k) \leq \sum_{k=2}^{n} (-1)^{k+1} \frac{\Delta_{a,k}}{k} + \Delta_{a,k}\), where \(\Delta_{a,k} = (2k + 1)\sqrt{a \ln n}\).

Proofs of these facts are omitted from this extended abstract.

**Acknowledgements.** We thank Shmuel Zaks for motivating parts of this work, and Idit Keidar and Mikołaj Rotkiewicz for their comments.
References


Grzegorz Malewicz is at the Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA. E-mail: greg@cse.uconn.edu

Alexander Russell is on the faculty of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA. E-mail: acr@cse.uconn.edu

Alex A. Shvartsman is on the faculty of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA. He is also a Research Associate at the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. E-mail: alex@theory.lcs.mit.edu
Efficient Condition-Based Consensus

ACHOUR MOSTÉFAOUI  
IRISA, France

SERGIO RAJSBAUM  
Compaq Research Lab, USA

MICHEL RAYNAL  
IRISA, France

MATTHIEU ROY  
IRISA, France

Abstract

The condition-based approach for consensus solvability (that we have introduced in a previous paper, ACM STOC’01) consists in identifying sets of input vectors for which it is possible to design a protocol solving the consensus problem for $n$ processes despite the occurrence of up to $f$ process crashes. For each value of $f$ these conditions actually defines a hierarchy. This paper continues our investigation of this approach. It has two main contributions. It first show that it is possible to define conditions from very simple weight functions. Interestingly any weight function define an acceptable condition, i.e., a condition that allows to solve the consensus problem. The second contribution is an efficient protocol whose wait-free part has a number of shared memory accesses upper bounded by $O(n \log_2(f + 1))$ (interestingly, this upper bound is rarely attained).

Keywords

asynchronous distributed system, condition, consensus problem, early decision, fault-tolerance, input vector, shared memory

1 Introduction

Context of the study  The Consensus problem lies at the heart of many distributed computing problems one has to solve when designing reliable applications on top of unreliable distributed asynchronous systems. There is a large literature dedicated to studying theoretical and practical aspects of this problem, that can be informally stated in terms of three requirements. Each process proposes a value, and has to decide on a value (termination) such that there is a single decided
value (agreement), and the decided value is one of the proposed values (validity). One of the most fundamental impossibility results in distributed computing says that this apparently simple problem has actually no deterministic solution in an asynchronous system even if only one process may crash [8]. To circumvent this impossibility, known as FLP, two main approaches have been investigated. One of them consists of relaxing the requirements of the problem, by either allowing for probabilistic solutions (e.g., [3]), or for approximate solutions ($\varepsilon$-agreement [6], or $k$-set agreement [5]). Another approach consists of enriching the system with synchrony assumptions until they allow the problem to be solved [7]. This approach has been abstracted in the notion of unreliable failure detectors [4].

We have recently introduced a new condition-based approach to tackle the consensus problem [9]. This approach focuses on sets of input vectors that allow $n$ processes to solve the consensus problem despite up to $f$ process crashes, in a standard asynchronous model. Let an input vector be a size $n$ vector, whose $i$-th entry contains the value proposed by a process $p_i$. A condition (which involves the parameters $f$ and $n$) is a set of such vectors that can be proposed under normal operating conditions. This approach is interested in $f$-fault tolerant protocols that (1) solve consensus at least when such a condition holds, and (2) are always safe. Safe means that the protocol guarantees agreement, whether the proposed input vector is allowed by the condition or not. In addition, this approach is also concerned in protocols that make their “best effort” to terminate (for example, they should terminate in all failure-free executions). This is the best we can hope for, since the FLP impossibility result says we cannot require that a consensus protocol terminates always, for every input vector. But, by guaranteeing that safety is never violated, the hope is that such a protocol should be useful in applications. For example, let us consider the condition “more than a majority of the processes propose the same value.” It is not hard to see that consensus can be solved when the inputs satisfy this condition, when $f = 1$. It is plausible to imagine an application that in some real system satisfies this condition most of the time (only when something goes wrong, the processes proposals get evenly divided).

Previous work on the condition-based approach In [9], we characterized the conditions that accept a consensus protocol with the above properties. That is, we described a set of conditions, denoted here $C_f$, and proved that there is a consensus protocol for a condition $C$ if and only if $C \in C_f$. We presented two equivalent combinatorial descriptions of the class $C_f$, and described two natural conditions $C_1$ and $C_2$ in $C_f$ that might be useful in practice, and proved them to be maximal (in the sense they cannot contain more input vectors). Basically, $C_1$ includes the vectors that favor their greatest value, while $C_2$ is made up of the vectors that favor their most common value. The class $C_f$ is quite rich, since it includes every condition for which there exists a condition-based consensus protocol. The protocol that we have presented in [9] can be instantiated for each particular condition $C \in C_f$. It has the same step complexity, whatever the condition it is instanti-
ated with, namely $\mathcal{O}(n \log_2(f + 1))$ read/write shared memory operations in the wait-free part of each process.

Although a priori it could be that all conditions of $\mathcal{C}_f$ are equally difficult to solve, an interesting question is the following “Are there conditions that are more difficult to solve than others?” If this is the case, there are more efficient protocols, specially tailored for particular classes of conditions. In practice, one would be interested in identifying the simplest classes of conditions whose input vectors occur frequently, because such classes would perhaps have very efficient consensus protocols. This question has been partially answered in [10] where we have shown that $\mathcal{C}_f$ actually defines a hierarchy of classes of conditions, each one of some degree $d$ ($0 \leq d$), namely, $\mathcal{C}_f^{d+1} \subseteq \mathcal{C}_f^d \subseteq \cdots \subseteq \mathcal{C}_f^1 \subseteq \mathcal{C}_f^0$ , where $\mathcal{C}_f^0$ is the class of the weakest conditions.

We have also presented in [10] a condition-based consensus protocol that can be used for any condition of degree $d$, and shown that the value $(f - d)$ measures the “difficulty” of the class $\mathcal{C}_f^d$. More precisely, it is shown that the number of read/write operations that are executed by a consensus protocol is related to $d$. Roughly speaking, for any condition $C$ in the class $\mathcal{C}_f^d$, the number of read/write invocations of the wait-free part of the protocol is proportional to $n \log_2(f - d + 1)$. Hence, when we progress in the hierarchy from the largest class $\mathcal{C}_f = \mathcal{C}_f^0$, there are more and more efficient consensus protocols, until the class $\mathcal{C}_f^{d+1}$.

Content of the paper This paper continues our study of the condition-based approach. It has two main contributions. For lack of space, the proofs are in Appendix II and a third contribution of the paper is given in Appendix III.

First contribution. A generic form to express conditions is first presented. This generic expression is based on weights associated with the values that can be proposed by the processes. Whatever the weight function used to instantiate the generic formula we get, for any pair $(f, d)$, a condition that belongs to $\mathcal{C}_f^d$. Interestingly, and in addition to its simplicity, this generic formulation not only includes the two previous particular conditions that we have investigated in previous works [9, 10] (namely, $C_1$ and $C_2$), but also allows to define new conditions to solve the consensus problem. While there are conditions that cannot be derived from the generic expression, it seems that the generic expression captures a meaningful set of practically relevant conditions.

Second contribution. The second contribution is related to the efficiency of condition-based consensus protocols. As noticed previously, the protocol presented in [9] solves the consensus problem for any condition in $\mathcal{C}_f$, but requires $O(n \log_2(f + 1))$ read/write shared memory operations per process in its wait-free part. Differently, the protocol described in [10] costs only $O(n \log_2(f - d + 1)$.

\footnote{In a sequel paper [12] we showed that for $f < n/2$, there is a more efficient protocol whose wait-free part has no operation.}
read/write shared memory operations per process, but guarantees termination only when the actual input vector $I$ belongs to $C_f$. Hence, if $I \in C_f - C_f^d$, it is possible that the protocol tailored for $C_f^d$ does not terminate.

Hence, the following question: “Is it possible to design a protocol for the conditions in $C_f$, that, when compared to the protocol presented in [9], can allow processes to decide earlier and at a lower cost when the input vector belongs to $C_f - C_f^d$?” It appears that the answer to this question is “yes”. To answer it, the paper presents such an adaptive protocol. Only in the worst case, a process has to execute $O(n \log_2(f + 1))$ read/write shared memory operations (in its wait-free part). The actual number of such operations actually depends on the number of actual failures and their perception by the processes.

**Organization of the paper** The paper is made up of 5 sections. Section 2 introduces the computation model, while Section 3 presents the condition-based approach. Then, Section 4 proposes a generic formula to derive conditions, and shows that all its instances belong to $C_f$. Section 5 presents an adaptive condition-based protocol and proves its correctness. This section shows also an interesting relation between the weights used to define a condition and a critical parameter used by the protocol. Due to space limitations, proof are omitted, they can be found in [11].

## 2 Computation Model

The computation model is a standard asynchronous shared-memory system with $n$ ($n > 1$) processes, where at most $f$ ($0 \leq f < n$) processes can crash. The shared memory consists of single-writer, multi-reader atomic registers. The executions are assumed to be linearizable.(For details of this model see any standard distributed computing textbook.)

The shared memory is organized into arrays. The $j$-th entry of such a shared array $X[1..n]$ can be read by any processes $p_i$ with an operation `read($X[j]$)`. Only $p_i$ can write to the $i$-th component, $X[i]$, it uses the operation `write($v$, $X[i]$)` for this. In addition to the shared memory, each process has a local memory. The subindex $i$ is used to denote $p_i$’s local variables.

To simplify the notation we also consider the following non-primitive, non-atomic `collect` operation which can be invoked by any process $p_i$. It can only be applied to a whole array $X[1..n]$, and is an abbreviation for $\forall j: \text{do read($X[j]$)} \text{ enddo}$. Hence, it returns an array of values $[a_1, \ldots, a_n]$ such that $a_j$ is the value returned by `read($X[j]$)`.

When we count the number of operations executed by a process we only count the number of read/write operations that access the shared memory. Local operations are not considered.
3 The Condition-Based Approach

3.1 The Approach

In the consensus problem there is a set $\mathcal{V}$ of values that can be proposed by the processes. In an execution, every correct process $p_i$ proposes a value $v_i \in \mathcal{V}$ and all correct processes have to decide on the same value $v$, that has to be one of the proposed values. The proposed values in an execution are represented as an input vector, such that the $i$-th entry contains the value proposed by $p_i$, or $\perp$ if $p_i$ did not take any step in the execution. We usually denote with $I$ an input vector with all entries in $\mathcal{V}$, and with $J$ an input vector that may have some entries equal to $\perp$. If at most $f$ processes can crash, we consider only input vectors $J$ with at most $f$ entries equal to $\perp$, called views. Let $\mathcal{V}^m$ be the set of all possible input vectors with all entries in $\mathcal{V}$, and $\mathcal{V}^m_f$ the set of all possible vectors with at most $f$ entries equal to $\perp$. For $I \in \mathcal{V}^m$, let $I_f$ be the set of possible views, i.e., the set of all input vectors $J$ with at most $f$ entries equal to $\perp$, and such that $I$ agrees with $J$ in all the non-$\perp$ entries of $J$. For a set $C$, $C \subseteq \mathcal{V}^m$, let $C_f$ be the union of the $I_f$'s over all $I \in C$. Thus, in the consensus problem, every vector $J \in \mathcal{V}^m_f$ is a possible input vector.

The condition-based approach consists of considering subsets $C$ of $\mathcal{V}^m$, called conditions, that represent common input vectors in a particular distributed application. We are interested in conditions $C$ that, when satisfied (i.e., when the proposed input vector does belong to $C_f$), make the consensus problem solvable, despite up to $f$ process crashes. More precisely, we say that an $f$-fault tolerant protocol solves the consensus problem for a condition $C$ if in every execution whose input vector $J$ is in $\mathcal{V}^m_f$, the protocol satisfies the following properties:

- **P-Validity**: A decided value is one of the proposed values (consensus validity).
- **P-Agreement**: Two processes cannot decide different values (consensus agreement).
- **P-Best-Effort Termination**: If (1) $J \in C_f$ and no more than $f$ processes crash, or (2) all processes are correct, or (3) a process decides, then every correct process decides.

We use the following notations in the rest of the paper. For vectors $J_1, J_2 \in \mathcal{V}^m$, $J_1 \leq J_2$ if $\forall k : J_1[k] \neq \perp \Rightarrow J_1[k] = J_2[k]$. The expression $\#_x(J)$ denotes number of entries of $J$ whose value is $x$, with $x \in \mathcal{V} \cup \{\perp\}$.

3.2 Acceptability of a Condition

Given a condition $C$, a value of $f$ and a degree $d$, acceptability (is a combinatorial property that) defines the constraints $C$ has to satisfy in order the consensus problem can be solved for $C$. Operationally, it is defined in terms of a predicate $P$ and a function $S$ that have to satisfy some properties. (A combinatorial characterization of acceptability is given in [9, 10].) Those properties are related to the termination, validity and agreement of the protocol, respectively.
The intuition for the first property is the following. The predicate $P$ allows a process $p_i$ to test if a decision value can be computed from its view. Thus, $P$ returns true at least for all those input vectors $J$ such that $J \in I_f$ for $I \in C$.

- Property $T_{C \rightarrow P}$: $I \in C \Rightarrow \forall J \in I_f : P(J)$.

The second property is related to validity.

- Property $V_{P \rightarrow S}$: $\forall I \in C^m : \forall J \in I_f : P(J) \Rightarrow S(J) = \text{a non-}\bot \text{ value of } J$.

The next property concerns agreement. Given an input vector $J$, for any condition $d$ when $d$ increases from $0$ to $f$, the quantity $d$ measures the difficulty of the condition: The stronger the condition (i.e., when $d$ increases from $0$ to $f$), the more efficient the protocol (there is no additional gain when $d > f$). Hence, there is a tradeoff relating the degree of a condition and the cost of the associated consensus protocol.

### Definition 1
A condition $C$ is $(f,d)$-acceptable if there exist a predicate $P$ and a function $S$ satisfying the properties $T_{C \rightarrow P}$, $A_{P \rightarrow S}^{[d]}$, and $V_{P \rightarrow S}$ for $f$. An $(f,d)$-acceptable condition is denoted $C^{[d]}$. The class $C_f^{[d]}$ consists of all the $(f,d)$-acceptable conditions.

### 3.3 Summary of Previous Results

A main result of [9] is the following theorem, which states that (given $f$) $C_f = C_f^{[0]}$ is the largest set of conditions that allows to solve the consensus problem.

**Theorem 1** The consensus problem is solvable for $C$ iff $C$ is $(f,0)$-acceptable.

As indicated in the Introduction, the condition-based consensus protocol presented in [9] is of the most general form as it works with any condition $C \in C_f$. As noted, the cost of its wait free part is $O(n \log_2(f + 1))$ read/write shared memory operations per process.

The theorem that follows (stated and proved in [10]) shows that $C_f$ is actually made up of a strict hierarchy of conditions.

**Theorem 2** $\forall d \geq 0 : \cdots \subseteq C_f^{[d+1]} \subseteq C_f^{[d]} \subseteq \cdots \subseteq C_f^{[0]} = C_f$.

The consensus protocol presented in [9] assumes a fixed $d$ and works with any condition in $C_f^{[d]}$. As noted previously, its cost is only $O(n \log_2(f - d + 1))$ read/write shared memory operations per process. The quantity $(f - d)$ actually measures the difficulty of the condition: The stronger the condition (i.e., when $d$ increases from $0$ to $f$), the more efficient the protocol (there is no additional gain when $d > f$). Hence, there is a tradeoff relating the degree of a condition and the cost of the associated consensus protocol.
4 A Generic Formulation of Conditions

Given $d$ and $f$, this section introduces a generic formulation for conditions. It also proves that all the conditions $C^{[d]}$ derived from this formulation are $(f, d)$-acceptable, i.e., belong to $C^{[d]}_f$. A few conditions derived from the generic formulation are exhibited.

4.1 Weight-Based Definition of Conditions

Let $w$ be a function from $V$ to $\mathbb{R}^+$. It associates a positive weight $w(a)$ with each value $a$ that can be proposed. In order not to introduce cumbersome notations in the following, an input vector is sometimes considered as the set of the values it contains.

The idea of the generic weight-based formulation of a condition $C$ is to provide a simple way to distinguish a value that can be safely decided from a vector. Given a degree $d$, we define the condition $C^{[d]}$ to be the set of vectors $I$ that satisfy:

$$\forall a \in I, \forall b \in I \setminus \{a\} : (\#_a(I) w(a) - \#_b(I) w(b) > (f + d) \max(w(a), w(b)))$$

The intuition that underlies this definition is the following. For a value $a$ of an input vector $I$ to be decided (i.e., for $I$ to belong to the condition $C^{[d]}$), this value has to “bypass” any other value $b$ present in $I$, (1) despite up to $f$ process crashes, and (2) whatever the occurrence number of that “adversary” value $b$. This means that:

- The value $a$ has to be “present enough” despite the previous situations. This is expressed by the following constraint: $(\#_a(I) - (f + d)) \cdot w(a) > \#_b(I) \cdot w(b)$.
- The value $a$ has to be “distant enough” from any $b$ to be distinguishable. This is expressed by the following constraint: $\#_a(I) \cdot w(a) > (\#_b(I) + (f + d)) \cdot w(b)$.

The weights are used to represent the respective “power” of each value of $V$. The generic definition is simply the combination of the two previous constraints involving the weights and the parameter $f + d$.

The theorem that follows shows that any condition defined from the previous generic weight-based expression defines an $(f, d)$-acceptable condition, i.e., belongs to $C^{[d]}_f$.

**Theorem 3** \( \forall d \geq 0 \), any function $w$ from $V$ to $\mathbb{R}^+$ defines a condition $C^{[d]}$ in $C^{[d]}_f$. For the proof, see [11].

4.2 Examples of Conditions

We consider here three weight functions. They differ in the way they a priori favor values. Other weight functions can easily be defined. Each of these weight

---

2A value needs to have a positive weight to be decided. So, in the following, we consider only positive weights in order not to a priori discard values from being decided. If one is interested in preventing some values $x$ to be decided, the corresponding weight $w(x)$ has to be set to 0.
functions defines a condition pattern $C$ that combined with the degree $d$ provides the following hierarchy of conditions: $\cdots \subset C^{[d+1]} \subset C^{[d]} \cdots \subset C^{[0]}$.

**Favoring no value**  (Condition pattern C2) Let us first consider the *uniform weight* function: $\forall a \in \mathcal{V}': w(a) = 1$. This function a priori favors no value. The generic expression simplifies and becomes:

$$
(I \in C^{[d]}_2) \equiv (\exists a \in I : \forall b \in I \setminus \{a\} : \#_a(I) - \#_b(I) > (f + d)).
$$

As we can see, this is the condition pattern called C2 in [9, 10]. It favors the value that appears the most often in an input vector. To be decided, despite up to $f$ crashes and the presence of any other value $b$, the most common value $a$ has to appear $(f + d)$ times more than $b$. Hence, this weighting function defines the hierarchy of conditions: $\cdots \subset C^{[d+1]}_2 \subset C^{[d]}_2 \cdots \subset C^{[0]}_2$.

$S2(J)$ = the most common value of $J$, and $P2^{[d]}(J) \equiv (\#_{1st}(J) - \#_{2nd}(J) > f + d - \#_1(J))$ constitute $(f, d)$-acceptability parameters for $C^{[d]}_2$ ($\#_{1st}(J)$ and $\#_{2nd}(J)$ denote the occurrence number of the most common value and second most common value of a vector, respectively).

**Slightly favoring a single value**  (Condition pattern C3) Let us associate the same weight (namely, 1) to all values but one, e.g., a whose weight is 2. This is a new condition, not investigated before, that we call C3. It very slightly favors the distinguished value $a$. Increasing the weight of $a$, would favor it more and more when it appears in an input vector.

**Largely spacing out the favors**  (Condition pattern C1) Let $\mathcal{V}' = \{a_1, \ldots, a_p\}$, and let us assume that these values are ranked: $a_1 < a_2 < \cdots < a_{p-1} < a_p$. Moreover, let $w(a_i) = n^i$. This condition favors $a_p$ (the largest value in $\mathcal{V}'$). Then, if $a_p$ is not proposed it favors $a_{p-1}$ (the second largest value in $\mathcal{V}'$), etc.

As all weights are powers of $n$, we can simplify the generic formula. Let $a_i$ be the largest value that appears in a vector $I$, and $a_j$ another value in $I$ (hence $i > j$). The constraint part in the formula becomes: $\#_{a_i}(I) n^i - \#_{a_j}(I) n^j > (f + d) \max(n^i, n^j)$, which simplifies into $\#_{a_i}(I) - \#_{a_j}(I) (1/n^{j-i}) > (f + d)$. As $i - j > 1$ and $\#_{a_j}(I) < n$, we have $0 < \#_{a_j}(I) (1/n^{j-i}) < 1$. This allows to simplify once more and we get:

$$
(I \in C^{[d]}_1) \equiv (a_i = \max(I) \Rightarrow \#_{a_i}(I) > (f + d)).
$$

that is is the condition pattern C1 investigated in [9, 10]: a vector belongs to the condition $C^{[d]}_1$ if its greatest value appears more than $(f + d)$ times. $S1(J)$ = the largest value of $J$, and $P1^{[d]}(J) \equiv (a_i = \max(J) \Rightarrow \#_{a_i} > f + d - \#_1(J))$ constitute $(f, d)$-acceptability parameters for $C^{[d]}_1$ (assuming $\bot$ is smaller than any value of $\mathcal{V}'$).

**Remark** Interestingly, when $n > 3$, $|\mathcal{V}'| = 2$ and $f = 1$, we have $C^{[0]}_1 \cup C^{[0]}_2 = \mathcal{V}'$. This shows that the set of $(f, 0)$-acceptable conditions is not closed under union (due to the FLP impossibility result, $\mathcal{V}'$ is not acceptable).
5 An Efficient Condition-Based Protocol

5.1 Structure of the Protocol

The condition-based consensus protocols presented in [9, 10] are made of three parts. In their first part, a process builds a view of the input vector. This view includes at least \((n - f)\) non-\(\perp\) values. The second part is a wait-free part where each process tries to get a decision value without compromising the consensus safety requirement. The last part is the best effort part of the protocol: a process that has not decided looks for the value decided by another process (if any), or else waits until it knows all processes have deposited the value they propose in the shared memory.

As noticed previously, the cost of these protocols is measured by the number of shared memory accesses issued in their wait-free part: the protocol presented in [9] works with any condition \(C \in C_f\) and costs \(O(n \log_2(f + 1))\), while the protocol presented in [10] depends on \(d\): it is tailored for the conditions of \(C_f^{(d)}\) and costs only \(O(n \log_2(f - d + 1))\). Let us consider an input vector that belongs to \(C_f^{(d')}\). If \(d' \geq d\), both protocols work but the second is more efficient. If \(d' < d\), only the first protocol guarantees the termination property.

The aim of this section is the design of a protocol that, when \(C \in C_f\), always terminates, but whose wait-free part does not always cost \(O(n \log_2(f + 1))\). This protocol (described in Figure 1) has the three-part structure just described. A process \(p_i\) starts executing the protocol by invoking the function \(\text{Consensus}(v_i)\) where \(v_i\) is the value it proposes. It terminates when it executes the statement \(\text{return}\) (at line 5, 7 or 10) which provides it with the decided value \(r\). The three-parts of the protocol are:

- **Part 1** (lines 1-2): A process \(p_i\) first writes its input value \(v_i\) to the shared array \(V\) (initialized to \([\perp, \ldots, \perp]\)). Then \(p_i\) repeatedly reads \(V\) until at least \((n - f)\) processes (including itself) have written their input values in \(V\), from which it constructs its initial view \(J_i\), where \(J_i[j]\) is the input value of \(p_j\), or \(\perp\) if \(p_j\) has not yet written its input value.

- **Part 2** (lines 3-5): Now, \(p_i\) enters its wait-free condition-dependent protocol part. It uses the underlying \(\text{Decision\_Chasing}\) function which is the core of the protocol from safety and efficiency point of views. This abstraction returns the decided value if \(p_i\) can decide by itself, or \(\top\) if it cannot. Whatever the output \(w_i\) provided by \(\text{Decision\_Chasing}\), \(p_i\) writes it into the shared variable \(W[i]\) (line 4). The array \(W\) is initialized to \([\perp, \ldots, \perp]\). If \(w_i \neq \top\), \(p_i\) can decide by itself: this writing will help processes that cannot decide by themselves. If \(w_i = \top\), this writing informs the other processes that \(p_i\) cannot decide by itself but has deposited its proposed value. Then, if \(p_i\) can decide, it does it (line 5). Otherwise it proceeds...
to the best effort termination part.

- **Part 3** (lines 6-10): In this section, $p_i$ enters a loop to look for a decision value (i.e., a value different from $\bot, \top$) provided by another process $p_k$ in the shared variable $W[k]$. If, while waiting for a decision, $p_i$ discovers that every process has written a value to $W$, and no process can directly decide (all these values are $\top$), $p_i$ concludes that every process has deposited its initial value in the shared array $V$ in line 1. Then, $p_i$ reads $V$ (line 10) to get the full input vector, and proceed to decide according to a fixed deterministic rule $F$ (such as max).

```plaintext
Function Consensus($v_i$) return($r$):
(1) write($v_i, V[i]$);
(2) repeat $A \leftarrow$ collect($V$) until ($\#A \leq f$) endrepeat;
(3) $w_i \leftarrow$ Decision_Chasing($A$);
(4) write($w_i, W[i]$);
(5) case $w_i \neq \top$ then return($r = w_i$)
(6) $w_i = \top$ then repeat $X_i \leftarrow$ collect($W$);
(7) until ($\exists k : X_i[k] \neq \bot, \top$) then return($r = X_i[k]$) endif
(8) $[a_1, \ldots, a_k] \leftarrow$ collect($V$);
(9) return($r = F([a_1, \ldots, a_k])$
(11) endcase
```

Figure 1: Efficient Consensus-Based Protocol

### 5.2 Adapting Attiya-Rachman’s Synchronization Tree

The Decision_Chasing function used in Figure 1 relies on the traversal of a classifier/improver tree close to Attiya-Rachman’s tree defined in [2]. The next subsection will show how and under which conditions the traversal of this adapted tree can be appropriately shortened (according to the information currently at the process’s disposal) in order to save shared memory accesses. The aim of this binary tree is to allow the processes that call Decision_Chasing to improve their current view $J_i$ and to classify them according to the relation “$\leq$” defined on views.

**Structure and traversal of the classifier tree** The tree structure is statically defined. It is a full binary tree with $\lceil \log_2 m \rceil$ levels and $m$ leaves, where $m = \lceil f/2 \rceil + 1$ [2] ($m$ is assumed to be a power of 2. Otherwise, the tree is not fully balanced). It is shared by all the processes; they access it from the pointer root. A process traverses the tree from the root downwards (as we will see, it can stop the traversal before a leaf). Hence, each non-leaf vertex $v$ contains pointers to its children, namely, $v.left$ and $v.right$. 
To allow the processes to classify their views, each vertex $v$ of the tree is labeled with an integer interval: $[L(v), H(v)]$, and the labels are statically set to satisfy the following properties:

- $[L(root), H(root)] = [n - f, n]$,
- $L(v.left) = L(v)$, $H(v.left) = L(v.right) - 1$, $H(v.right) = H(v)$ for any non-leaf vertex $v$.

Hence, the intervals associated with the leaves of the tree form a partition of the interval associated with the root. Such a labeled tree, with $f = 2^p - 2$ (for some $p$), is depicted in Figure 2. An interval is trivial if its size is 1.

**Using the tree to improve the views** From the “dynamic” point of view, each vertex $v$ of the tree contains an array $v.R$ of $n$ vectors. The vector $v.R[i]$ is a shared variable initialized to $[\bot, \ldots, \bot]$ that can atomically be written by $p_i$ (to store its current view) and read by any process. Let the size of a view $J$ be $|J| = n - \#_\bot(J)$ (number of positions of $J$ with a non-$\bot$ value).

![Figure 2: A Classifier/Improver Tree](image-url)

The basic step of the traversal of the tree by a process $p_i$ is described in Figure 3. As indicated, a process traverses it from the root downwards. When it visits the vertex $v$, $p_i$ first deposits its current view in $v.R[i]$ (line 1). Then, $p_i$ reads the other views $v.R[j]$ already deposited in $v.R$ by other processes $p_j$ (line 2). According to what it has read, it progresses (classifies) to the right son or the left son of $v$. The notation $J_i$ (used at lines 3 and 5), for views $J_i \in I_f$ denotes the view $J$ that has a non-$\bot$ value $v = J_i[j]$ in a position $j$ if at least for one $i$, $J_i[j] = v$. If $p_i$ knows more than $H(v.left)$ entries different from $\bot$, it adopts the current content of $v.R$ as its current view and progresses to the right son of $v$ (lines 4-5). In the other case, it keeps its previous view $J_i$ and progresses to the left son of $v$ (line 6).
Properties associated with the tree Let us denote \( p_i \) \( \in \text{visited}(v) \) the execution (if any) of \( v.\text{Classifier}_\text{Improver} \) by \( p_i \). Moreover, let \( J_{i,v} \) denote the actual value of the corresponding input parameter, while \( \text{dir}_{i,v} \) denotes the value returned in the output parameter \( \text{dir}_i \) (i.e., at the end of the visit of \( v \) by \( p_i \)). The lemma that follows, proved in [10], states the main properties of the tree traversal.

**Lemma 1** Let us assume that the initial input view \( J_I \) of every \( p_i \) that calls \( \text{Decision}_\text{Chasing} \) belongs to \( I \) for some input vector \( I \). For every vertex \( v \) and every process \( p_i \) we have the following.

1. \( L(v) \leq |J_{i,v}| \leq H(v) \) when \( p_i \in \text{visited}(v) \).
2. \( |\cup_{p_i \in \text{visited}(v)} \{J_{i,v}\}| \leq H(v) \).
3. \( J_{i,v} < J_{j,u} \) whenever \( H(v) < L(u) \land p_i \in \text{visited}(v) \land p_j \in \text{visited}(u) \).
4. \( J_{i,v} = J_{j,v} \) whenever \( L(v) = H(v) \land p_i, p_j \in \text{visited}(v) \).
5. The cost of \( v.\text{Classifier}_\text{Improver} \) for a process is upper bounded by \((2n + 1)\) shared memory read/write operations\(^3\).

5.3 Shortening the Tree Traversal

In the protocol presented in [9] a process traverses the previous tree once, from the root to a leaf. Hence, its cost (number of shared memory operations) lies between \((n + 1)D\) and \((2n + 1)D\), where \( D \) is the depth of the tree, namely, \( D = \lceil \log_2([f/2] + 1) \rceil \). In the protocol tailored for the stronger conditions of \( C'_{d,f}^{t,d} \) [10], a process also traverses the tree from the root to a leaf, but here the tree has a shorter depth, namely, \( D' = \lceil \log_2([ (f - d)/2 ] + 1) \rceil \).

The idea that underlies the design of the \( \text{Decision}_\text{Chasing} \) function described in Figure 4 is to allow a process to stop its tree traversal before having

---

\(^3\)Those are the accesses to the array \( R \) associated with the vertex \( v \): one write, one collect, plus possibly one more collect.
joined a leaf. This will obviously decrease the length of the path traversed by the process and consequently allow it to decide early thereby reducing the cost of its wait-free part. The difficulty in attaining this goal consists in ensuring that, whatever the vertices at which two processes $p_i$ and $p_j$ stop their tree traversal, they will decide the same non-$\bot$ value: the consensus agreement property must be guaranteed whatever parts of the tree are traversed by processes. Moreover, given a condition $C \in C_J$, the protocol must terminate at least when $I \in C$.

**Underlying idea of the protocol**  Let $C \subseteq C_J$. As we have seen, $C$ gives rise to a hierarchy of conditions $\cdots \subseteq C^{d+1} \subseteq C^d \subseteq \cdots \subseteq C^0 = C$. Let $P^d$ be a predicate associated with $C^d$.

When a process $p_i$ visits a vertex $v$ and gets $J$ as current view, the idea is to allow it to try to benefit from the fact that $J$ can belong to some $I_f'$ such that $I_f' \subseteq C^d$, in order to stop its tree traversal. Only in the worst case a process has to traverse the tree from the root to a leaf. In order the agreement property be never violated by processes stopping at arbitrary vertices, the protocol requires that the predicates $P^d$ satisfy monotonicity properties.

**Properties on the sequence of predicates**  When it visits a vertex $v$, a process has first to compute a value $d$ to test if $P^d(J)$ holds ($J$ being its current view). The determination of $d$ is crucial for the protocol correctness: $d$ has to be “as small as possible” to allow early tree exit, but large enough to guarantee agreement.

Let us consider two processes that exist the tree at $v1$ and $v2$ with $J1$ and $J2$, respectively. This means that both $P^{d1}(J1)$ and $P^{d2}(J2)$ hold. To ensure that we also have $S(J1) = S(J2)$, the protocol requires that the predicates $P^d$ satisfy monotonicity properties.

- **Decreasing Degree Monotonicity:**
  
  $M_{dd} \equiv (\forall d > 0, \forall J \in V^d_P, \forall J' : P^d(J) \Rightarrow P^{d-1}(J))$.

- **Increasing Vector Monotonicity:**
  
  $M_{iv} \equiv (\forall d \geq 0, \forall J \in V^d_P, \forall d', \forall J' \geq J : (P^d(J) \land \text{dist}(J, J') \leq \frac{d-d'}{\alpha(J')}) \Rightarrow P^{d'}(J'))$.

Let $J$ be a view such that $P^d(J)$ holds. This means that there exists $I \in C^d$ with $J \in I_f$. As $C^d \subseteq C^{d-1}$, it follows that $I \in C^{d-1}$. Hence $P^{d-1}(J)$ holds, and the sequence of predicates $P^d$ of any condition pattern satisfies $M_{dd}$.

Differently from $M_{dd}$, the property $M_{iv}$ involves a parameter $\alpha$ for each $J$. If follows that for the sequence of predicates $P^d$ to satisfy $M_{iv}$, the value $\alpha(J)$ associated with each view $J$ actually depends on the condition pattern (this issue is addressed in Section 5.4). The intuition that underlies the $M_{iv}$ property is the following. For each $J$, there is a “disc of augmented vectors” $J'$, centered at $J$ and with radius $\frac{d-d'}{\alpha(J')}$. The property $M_{iv}$ requires that all those augmented vectors satisfy $P^d$. As we start from $d$ and $J$, and proceed to $d' \leq d$ and $J' \geq J$, $M_{iv}$ states
a tradeoff relating how much an increase in the number of non-⊥ values in a view (from $J$ to $J'$) affects a degree reduction (from $d$ to $d'$) when we want to have both $P^{d[J]}$ and $P^{d'[J']}$. 

Let $p_i$ and $p_j$ be two processes with current views $J$ and $J'$, respectively. Let us consider the extreme case where $d' = 0$ and $d = \alpha(J) \ dist(J, J')$. If $P^{d[J]}$ holds (allowing $p_i$ to early decide), the property $M_{iv}$ allows $p_j$ to also decide. So, when $p_i$ is visiting the vertex $v$ and has $J$ as current view, as it does not know the view $J'$, it has to use some value to consistently approximate $dist(J, J')$. To that aim, it considers the value $\#_{\bot}(J) + n - L(v) - f$. As $d$ cannot be negative, it actually uses $\max(0, \alpha(J) \times (\#_{\bot}(J) + n - L(v) - f))$ to define the value of $d$ it uses to compute $P^{d[J]}$. The proof (see [12]) will show that this heuristic value is consistent.

**The core of the protocol** The Decision\_Chasing function (Figure 4) assumes that the sequence of predicates $P^{d[J]}$ associated with the condition $C$ satisfies $M_{id}$ and $M_{iv}$.

```plaintext
Function Decision\_Chasing(J) return(w_i):
(1) current_i ← J; v ← root; prev_v_val_i ← ∞; prev_nb_of_bot_i ← (f + 1);
(2) while v is not a leaf do
(3) (current_i, direction_i) ← v:\ Classifier\_Improver\(current_i\);
(4) let $d = \max(0, \alpha(current_i) \times (\#_{\bot}(current_i) + n - L(v) - f));$
(5) if $[prev_v_val_i \neq d] \lor [prev_nb_of_bot_i \neq \#_{\bot}(current_i)]$
(6) then if $P^{d[current_i]}$ then return($w_i = S(current_i)$) endif;
(7) prev_v_val_i ← d; prev_nb_of_bot_i ← $\#_{\bot}(current_i)$
(8) endif;
(9) if direction_i = right then $v ← v\\right$ else $v ← v\\left$ endif
(10) endwhile;
(11) if $P^{d[current_i]}$ then return ($w_i = S(current_i)$) else return ($w_i = \bot$) endif
```

Figure 4: The Decision\_Chasing Function

A process $p_i$ executing Decision\_Chasing first initializes local variables (line 1): $current_i$ represents its current view, $v$ the vertex it is about to visit (initially root), $prev_v_val_i$ the previous value of the parameter $d$, and $prev_nb_of_bot_i$ the occurrence number of the ⊥ value in its previous view.

Then $p_i$ enters the tree traversal (lines 2-10). It first visits the vertex $v$ (line 3), computes the current values of $\alpha(current_i)$ and $d$ from its context (current values of $current_i$ and $v$). If something has changed (line 5) with respect to the previous predicate evaluation (either the value of $d$ or $\#_{\bot}(current_i)$), then $p_i$ tests $P^{d[current_i]}$, and if true, stops the tree traversal and considers the value $S(current_i)$ (line 6). Otherwise (line 9), $p_i$ updates the relevant local loop variables, and progresses to the next level of the tree in the direction indicated by the previous call to Classifier\_Improver.

Finally, if a process has not stopped during the tree traversal (it has joined a leaf), we are in the case where there is no possibility to have an early decision.
So, \( p_i \) checks \( P^{[i]}(current_i) \) to see if it can decide. If it cannot, it sets \( w_i \) to \( \top \).

**Theorem 4** The Consensus protocol described in Figure 1 satisfies the P-Validity, P-Agreement and P-Best_Effort_Termination properties described in Section 3.1. Moreover, the number of shared memory accesses of its wait-free part is upper bounded by \( O(n \log_2(f + 1)) \) for each process. (Proof: see [11].)

**Example** Let us consider the following example to illustrate the early decision possibility offered by the Decision_chasing function. Let \( V = \{a, b, \ldots\} \), \( n = 6 \), \( f = 1 \). The condition considered is \( C2 \) defined by the uniform weight function (Section 4.2). This condition provides \( \alpha(J) = 2, \forall J \in V \) (See below).

Let us consider the input vector \( I = [a, a, a, a, b, b] \). As \( \#_a(I) - \#_b(I) > f \), it follows that \( I \in C2^{[0]} \). Moreover, it is easy to see that \( I \notin C2^{[d]} \) for \( d > 0 \).

Let \( current_i = [a, a, a, a, \bot, b] \) be the view of \( p_i \) after its first call to Classifier_improver at line 3. As \( v = root \) and \( L(root) = n - f \), we get \( d = 2 \) at line 4. Hence \( p_i \) evaluates \( P^{[d]}([a, a, a, a, \bot, b]) = (\#_a(current_i) - \#_b(current_i) > f + d - \#_a(current_i)) \) which is true. Consequently, \( p_i \) stops its tree traversal and exits with the value \( a \) (line 6). The fact that \( P^{[d]}([a, a, a, a, \bot, b]) \) holds means that there is some \( I' \in C2^{[2]} \) such that \( J \in I' \) (e.g., \([a, a, a, a, a, \bot] \) is such an \( I' \).

### 5.4 From a Condition to an \( \alpha \) Function

This section defines an \( \alpha \) function that allows predicates \( P \) associated with a condition obtained from a weight function (Section 4.2) to satisfy \( M_w \), i.e., \( \forall d \geq 0, \forall J \in V^d, \forall J' \geq J : (P^{[d]}(J) \land d' \leq d - \alpha(J) \text{dist}(J, J')) \Rightarrow P^{[d']}(J') \).

**Theorem 5** Let \( C \) be a condition defined from a weight function \( w \), with the following acceptability parameters (defined in Theorem 3):

\[
P^{[d]}(J) \equiv \exists a \in J, \forall b \neq a \in J, \quad (#a(J) + #b(J))w(a) - #b(J)w(b) > (f + d)\max(w(a), w(b));
\]

\[
S(J) = a \text{ such that } #a(J)w(a) = \max\{#b(J)w(b) : b \in J\}.
\]

Let \( J \in V^d, a = S(J), \) and \( M \) be the maximal weight associated with a value of \( V \). Let \( \alpha \) be a function from \( V^d \) to \( \mathbb{R}^+ \) defined as follows:

- \( \alpha(J) = \max_{a \in J, b \neq a} \left( \frac{w(a) + w(b)}{\max(w(a), w(b))} \right) \), if \( \exists x \in J : w(x) = M \).
- \( \alpha(J) = (n - f) \), otherwise.

The sequence of predicates \( P^{[d]} \) satisfies the monotonicity property \( M_w \) with this function \( \alpha \). (Proof: see [11].)

Table 1 defines the \( \alpha \) functions associated with the conditions \( C1, C2 \) and \( C3 \) defined in Section 4.2 (\( M \) denotes the maximal weight). As we can see, every condition provides an \( \alpha(J) \) that is always \( \geq 1 \).
Table 1: Values of α(J) for the three Conditions

<table>
<thead>
<tr>
<th>Condition</th>
<th>Condition</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>∃ x ∈ J : w(x) = M</td>
<td>∀ x ∈ J : w(x) &lt; M</td>
</tr>
<tr>
<td>C1</td>
<td>α(J) = 1 + 1/n</td>
<td>α(J) = (n – f)</td>
</tr>
<tr>
<td>C2</td>
<td>α(J) = 2</td>
<td>Cannot appear</td>
</tr>
<tr>
<td>C3</td>
<td>α(J) = 2 or 1.5 (acc. to J)</td>
<td>α(J) = (n – f)</td>
</tr>
</tbody>
</table>

References


Achour Mostéfaoui is Assistant Professor at the Computer Science Department of the University of Rennes, Campus de Beaulieu, 35042 Rennes, France. E-mail: achour@irisa.fr

Sergio Rajbaum is with the Cambridge Research Lab of Compaq, One Kendall Square Build. 700, Cambridge, MA 02139, and Professor at the Instituto de Matemáticas, UNAM, Ciudad Universitaria, D.F. 04510, Mexico. E-mail: rajsbaum@compaq.com

Michel Raynal is Professor at the Computer Science Department of the University of Rennes, Campus de Beaulieu, 35042 Rennes, France. E-mail: raynal@irisa.fr

Matthieu Roy is a PhD student at the Computer Science Department of the University of Rennes, Campus de Beaulieu, 35042 Rennes, France. E-mail: mroy@irisa.fr
Some Upper Bounds on Expected Agreement Time of a Probabilistic Local Majority Polling Game *

T. NAKATA
Fukuoka University of Education, Japan

M. YAMASHITA
Kyushu University, Japan

Abstract

This paper investigates the expected agreement time of a probabilistic polling game on a connected graph. Given a connected graph $G$ with an assignment of a value in $\{0, 1\}$ to each vertex, we consider a polling game on $G$ that repeats the following $k$-polling forever: $k$ randomly chosen vertices $v$, synchronously and independently, update their values to $\varepsilon \in \{0, 1\}$ with probability $n(\varepsilon) / |\Gamma(v)|$, where $\Gamma(v)$ is the set of neighbors of $v$, including $v$ itself, and $n(\varepsilon)$ is the number of vertices in $\Gamma(v)$ whose current value is $\varepsilon$. Given an initial value assignment, we give some upper bounds on the expected number of $k$-pollings necessary for the system to reach a global state in which all vertices have the same value, by using a martingale theory. We, in particular, give a good upper bound when $G$ is complete. Note that some special cases are known as Wright-Fisher’s and Moran’s models in population genetics.

Keywords
agreement problem, local majority polling, graph theory, Markov chain, martingale

1 Introduction

Let $G(V, E)$ be a connected undirected graph with order $|V| = n < \infty$. We assign, to each vertex $v \in V$, a value $\xi(v) \in \{0, 1\}$. A global state is the set of values that the vertices have and is denoted by $\xi = (\xi(v_1), \ldots, \xi(v_n)) \in \Xi = \{0, 1\}^V$. Let $\Gamma(v) = \{v\} \cup \{u \in V : \{u, v\} \in E\}$ be the set of neighbors of $v$, including $v$.

*This work has been supported in part by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Grant-in-Aid for Encouragement of Young Scientists, No.12740060.
itself. The number of vertices from \( \Gamma(v) \) having value \( \epsilon \in \{0, 1\} \) at \( \xi \) is denoted by \( n_\xi(v, \epsilon) \), i.e., \( n_\xi(v, \epsilon) = |\{w \in \Gamma(v) : \xi(w) = \epsilon\}| \) and

\[
n_\xi(v, 0) + n_\xi(v, 1) = |\Gamma(v)| \quad \text{for } v \in V. \tag{1}
\]

This paper discusses a probabilistic polling game on \( G \) defined as a repetitive execution of the following probabilistic procedure named \( k \)-polling: Let \( \xi = (\xi(v))_{v \in V} \in \Xi \) be the current global state. Then \( k \) randomly chosen vertices \( v \in V \), simultaneously and independently, update their values \( \xi(v) \) to \( \epsilon \in \{0, 1\} \) with probability \( n_\xi(v, \epsilon)/|\Gamma(v)| \). The probabilistic polling game defined by \( n \)-polling was first introduced by Peleg in connection with distributed agreement and other related problems; we regard the game as an agreement process, where an agreement is achieved when all vertices have the same value [8]. Recently Hassin and Peleg [4] and Nakata et al. [6] independently studied the game. Nakata et al. [6] discussed the game defined by \( k \)-polling for general \( k \), while Hassin and Peleg [4] concentrated on \( n \)-polling. Another slight difference is that in [4], the set of neighbors \( \Gamma(v) \) excludes \( v \).

The probabilistic polling game defined by \( k \)-polling is naturally formulated in terms of \( \Xi \)-valued Markov chain \( \{X_t^{(k)}\}_{t=0,1,...} \), where \( X_t^{(k)} = (X_t^{(k)}(v))_{v \in V} \) whose component \( X_t^{(k)}(v) \) is the value of \( v \) at time \( t \). We consider the probability space \( (\Omega, \mathcal{F}, \mathbf{P}_\xi) \) with an initial state \( \xi \in \Xi \), i.e., \( \mathbf{P}_\xi\{X_0^{(k)} = \xi\} = 1 \). For \( k \)-polling, the transition probability from \( \xi \) to \( \eta \) is given as follows:

\[
 p_t(\xi, \eta) = p_t(\xi, \eta^A) = \frac{1}{\binom{n}{k}} \sum_{U : A \subseteq U \subseteq \text{Sub}_k(V)} \prod_{v \in U} \frac{N_\xi(v, \eta^A(v))}{|\Gamma(v)|}, \tag{2}
\]

where \( \text{Sub}_k(V) \) denotes the set of all \( k \)-(sub)sets \( X \) of \( V \), i.e., \( \text{Sub}_k(V) = \{X \subseteq V : |X| = k\} \) and

\[
 \eta = \eta^A(v) = \begin{cases} 
 \xi(v), & \text{if } v \notin A, \\
 1 - \xi(v), & \text{if } v \in A.
 \end{cases}
\]

The following is a partial list of problems concerning this Markov chain:

(I) Except for two trivial absorbing states \( 0 = (0, \cdots, 0) \) and \( 1 = (1, \cdots, 1) \), are all states \( \xi \in \Xi \) transitive?

(II) If the answer for (I) is YES, for a given initial state \( \xi \), calculate the absorbing probability to \( 0/1 \), i.e., the probability that all vertices agree on value \( 0/1 \).

(III) Estimate the agreement time \( T \) necessary for the system to reach an absorbing state, where \( T = \inf\{t \in \mathbb{N} : X_t^{(k)} \in \{0, 1\}\} \).
Under our definition of $\Gamma/v$, i.e., $v \in \Gamma/v$, the answer for (I) is obviously YES. Under the definition of $\Gamma/v$ in [4], the answer for (I) is YES if $G$ is bipartite. Hassin and Peleg, hence, studied non-bipartite graphs in [4].

As for (II), letting $\text{Absorb}_k(\xi, 1)$ be the absorbing probability from $\xi$ to 1, [4, 6] show that

$$\text{Absorb}_k(\xi, 1) = \sum_{v \in V, \xi(v) = 1} |\Gamma(v)| \sum_{w \in V} |\Gamma(w)|$$

for $k = 1, \ldots, n$. (3)

The next theorem plays an essential role in proving Eq. (3).

**Theorem 1** ([4, 6]) Let $\{F_t\}_{t=0,1,\ldots}$ be the filtration of Markov chain $\{X_t^{(k)}\}_{t=0,1,\ldots}$. Then $(x_t, F_t)$ is a martingale. That is for any $t$

$$E_\xi [x_{t+1} | F_t] = x_t \text{ P}_\xi \text{-a.s.},$$

where

$$x_t = x_t^{(k)} = \sum_{v \in V, X_t^{(k)}(v) = 1} |\Gamma(v)|$$

and $E_\xi [\cdot]$ is the expectation with initial state $\xi$.

We will sometimes omit index $k$ from $x_t^{(k)}$ in the following, when it is clear from the context. By virtue of Theorem 1, since $T = \inf\{t : X_t^{(k)} \in \{0, 1\}\}$ has the stopping time property, we apply, to $x_t$, the optional stopping theorem [5, Corollary 3-16] to show

$$\text{Absorb}_k(\xi, 1) \cdot \sum_{v \in V} |\Gamma(v)| + \text{Absorb}_k(\xi, 0) \cdot 0 = E_\xi [x_T] = E_\xi [x_0] = \sum_{v \in V, \xi(v) = 1} |\Gamma(v)|.$$  

Thus we have Eq. (3).

As for (III), let $E_\xi [T]$ be the expected agreement time necessary for the system to reach 0/1 from $\xi$. Then [6, Theorem 7] states that $E_\xi [T]$ satisfies the following difference equations:

$$E_\xi [T] = \sum_{\eta \in \Xi} p_{\xi, \eta} E_{\eta} [T] + 1, \quad E_\xi [T] |_{E_\xi [T] = E_{0} [T] = E_{1} [T] = 0}. (5)$$

Thus $E_\xi [T]$ is computable by solving a set of simultaneous linear equations with $2^n - 2$ variables, but obtaining its explicit form seems to be difficult.

Hassin and Peleg [4, Theorem 2] proposed an upper bound on the expected agreement time for $n$-polling, by using another Markov chain $H$ with state space $V$.

**Theorem 2** ([4, Theorem 2]) If the Markov chain $H$ is reversible, then the expected agreement time for $n$-polling is $O(M \log n)$, where $M$ is the maximal meeting time for two random walks on $G$. 
Since Markov chain \( H \) is always reversible under the current setting, \( O(n^3 \ln n) \) is an upper bound on the expected agreement time for \( n \)-polling (assuming the definition neighbors in [4]), because the meeting time is bounded by \( O(n^3) \).

In this paper, we give an explicit upper bound on the expected agreement time for \( k \)-polling (assuming the definition neighbors in this paper). The bound depends on initial global state \( \xi \) as well as \( G \), so that we can obtain, for some initial global states, a better bound than a one depending only on \( G \), such as \( O(n^3 \ln n) \) in [4].

The rest of the paper is organized as follows: In Section 2, we first discuss general connected graphs and then complete graphs. The probabilistic polling games defined by 1- and \( n \)-pollings on a complete graph are respectively known as Moran’s and Wright-Fisher’s models in population genetics [1]. Thus \( k \)-polling on the complete graph is a natural interpolation of the above two models. Concluding remarks are given in Section 3.

2 Upper Bounds on the Expected Agreement Time

2.1 General Connected Graphs

Let us recall an integer valued stochastic process \( x_t \) from \( \Xi \)-valued Markov chain \( X_t \) discussed in Theorem 1. Let

\[
y_t^{(k)} = x_{t+1}^{(k)} - x_t^{(k)},
\]

which denotes the “efficiency” of polling at \( t \). We may omit index \( k \) from \( y_t^{(k)} \), whenever it is clear from the context.

**Lemma 1** The following three statements hold for \( x_t \) and \( y_t \).

(i) \( y_{t+1} = \sum_{v \in V: X_t^{(k)}(v) = 0, X_{t+1}^{(k)}(v) = 1} |\Gamma(v)| - \sum_{w \in V: X_t^{(k)}(w) = 0, X_{t+1}^{(k)}(w) = 1} |\Gamma(w)| \) \( \mathbb{P}_\xi \)-a.s.,

(ii) \( E_\xi[y_t] = 0 \) for any \( t \), and \( y_{t+1} \) is independent of \( \mathcal{F}_t \).

(iii) \( (x_t^2 - \sum_{i=1}^t \sigma_i^2; \mathcal{F}_t) \) is martingale, where \( \sigma_i^2 = E_\xi[|y_i|^2] \).

**Proof.** Item (i) is obvious since the following equality holds:

\[
\{ v \in V : X_t^{(k)}(v) = 1 \} \cup \{ v \in V : X_t^{(k)}(v) = 0, X_{t+1}^{(k)}(v) = 1 \} \\
= \{ v \in V : X_t^{(k)}(v) = 1 \} \cup \{ v \in V : X_t^{(k)}(v) = 1, X_{t+1}^{(k)}(v) = 0 \}.
\]

Since \( (x_t, \mathcal{F}_t) \) is martingale, \( E_\xi[x_t] = E_\xi[x_0] = \sum_{v \in V: \xi(v) = 1} |\Gamma(v)| \) for any \( t \). Hence by definition \( E_\xi[y_t] = 0 \) for any \( t \). Note that the law of polling is independent of time \( t \). Therefore \( y_{t+1} \) is independent of \( \mathcal{F}_t \). Consequently Item (ii) holds.
As for Item (iii), we have
\[ E_x \left[ x_{t+1}^2 - \sum_{i=1}^{t+1} \sigma_i^2 \right] = E_x \left[ x_t^2 + 2x_t y_{t+1} + y_{t+1}^2 - \sum_{i=1}^{t} \sigma_i^2 - \sigma_{t+1}^2 \right] = x_t^2 - \sum_{i=1}^{t} \sigma_i^2 + 2x_t E_x[y_{t+1} | f_t] + E_x[y_{t+1}^2] - \sigma_{t+1}^2 = x_t^2 - \sum_{i=1}^{t} \sigma_i^2. \]

Note that the last equality is due to Item (ii).

Since the variation \( \sigma_i^2 \) of random variable \( y_i^{(k)} \) depends on \( k \), it is sometimes denoted by \( \sigma_i^2(k) \). It represents the "speed" of \( k \)-polling; if \( \sigma_i^2(k) \) is small for any \( t \) then the expected agreement time is large. Put the non-trivial minimum of the variation \( \sigma_{\min}^2 = \sigma_{\min}^2(k) = \min_{v \in V} \min_{1 \leq t \leq T(v)} \sigma_t^2(k) > 0 \). Now we estimate it.

**Proposition 1** Let \( d \) be the minimum degree of \( G \), i.e., \( d = \min_{v \in V} |\Gamma(v)| - 1 \). Then
\[ \sigma_{\min}^2 = \sigma_{\min}^2(k) \geq \frac{k(2n-k-1)d}{n(n-1)}. \]

**Proof.** By Item (i) of Lemma 1,
\[ \sigma_{\min}^2(k) = \min_{\xi \in \hat{\Xi}} E_x \left[ \left( \sum_{\xi(v)=1,\xi(w)=0} |\Gamma(v)| - \sum_{\xi(w)=0,\xi(w)=1} |\Gamma(w)| \right)^2 \right], \]
where \( \hat{\Xi} = \Xi \setminus \{0,1\} \). By the definition of transition probability of Eq. (2),
\[ \sigma_{\min}^2(k) = \min_{\xi \in \hat{\Xi}} \left\{ \frac{1}{|\Gamma(v)|^2} \sum_{U \subseteq \text{Sub}_U(V)} \sum_{A \subseteq U} \left( \sum_{\xi(v)=1,\xi(w)=0} |\Gamma(v)| - \sum_{\xi(w)=0,\xi(w)=1} |\Gamma(w)| \right)^2 \right\}. \]

For a fixed \( \xi \), let
\[ V' = V'_{\xi} = \{ v \in V : w \in \Gamma(v) \text{ s.t. } \xi(v) \neq \xi(w) \}. \tag{7} \]

Then since \( \xi \in \hat{\Xi} \), we have
\[ |V'| \geq 2, \quad N_{\xi(v,0),N_{\xi(v,1)} \geq 1}, \quad N_{\xi(v,0),N_{\xi(v,1)} = |\Gamma(v)| \geq d + 1 \quad v \in V'. \tag{8} \]
Thereby we deduce that

\[ \sigma_{\min}(k) \leq \min_{\xi \in \mathbb{Z}} \left\{ \left( \sum_{U \subseteq \text{Sub}(V) \mid U \cap V' \neq \emptyset} \sum_{A \subseteq U} \left( \sum_{\xi(v) = 1, \xi(w) = 0} |\Gamma(v)| - \sum_{\xi(w) = 0, \xi(w) = 1} |\Gamma(w)| \right) \prod_{u \in U} \frac{N_2(u, \xi^A(u))}{|\Gamma(u)|} \right\} \right. \]

For each \( U \in \text{Sub}(V) \) satisfying \( U \cap V' \neq \emptyset \), we put

\[ I_2(U) = \sum_{A \subseteq U} \left( \sum_{\xi(v) = 1, \xi(w) = 0} |\Gamma(v)| - \sum_{\xi(w) = 0, \xi(w) = 1} |\Gamma(w)| \right) \prod_{u \in U} \frac{N_2(u, \xi^A(u))}{|\Gamma(u)|} \]

Letting \( v_s \in U \cap V' \), we have that for \( A = A_1 \cup A_2 \)

\[ I_2(U) = \sum_{A_1 \subseteq U \setminus \{v_s\}} \prod_{v \in U \setminus \{v_s\}} N_2(u, \xi^A(u)) \sum_{A_2 \subseteq \{v_s\}} \frac{N_2(v_s, \xi^{A_2}(v_s))}{|\Gamma(v_s)|} \times \left( \sum_{\xi(v) = 1, \xi^2(w) = 0} |\Gamma(v)| - \sum_{\xi(w) = 0, \xi(w) = 1} |\Gamma(w)| \right)^2 \]

Without loss of generality, assume \( \xi(v_s) = 1 \). Then since \( A_2 = \{v_s\} \) or \( \emptyset \), we have the estimation of the last half of \( I_2(U) \) as

\[ \sum_{A_2 \subseteq \{v_s\}} \frac{N_2(v_s, \xi^{A_2}(v_s))}{|\Gamma(v_s)|} \left( \sum_{\xi(v) = 1, \xi^2(w) = 0} |\Gamma(v)| - \sum_{\xi(w) = 0, \xi(w) = 1} |\Gamma(w)| \right)^2 \]

(9)

\[ = \frac{N_2(v_s, 0)}{|\Gamma(v_s)|} \left( \sum_{\xi(v) = 1, \xi^2(w) = 0} |\Gamma(v)| - \sum_{\xi(w) = 0, \xi(w) = 1} |\Gamma(w)| + |\Gamma(v_s)| \right)^2 \]

(9)

\[ + \frac{N_2(v_s, 1)}{|\Gamma(v_s)|} \left( \sum_{\xi(v) = 1, \xi^2(w) = 0} |\Gamma(v)| - \sum_{\xi(w) = 0, \xi(w) = 1} |\Gamma(w)| \right)^2 \]

In general, for any real number \( z \) and \( r \geq 2, 1 \leq w \leq r \),

\[ \frac{w}{r}(z + r)^2 + \left( 1 - \frac{w}{r} \right) z^2 = z^2 + 2wr + wr = (z + w)^2 + w(r - w) \geq w(r - w). \]

By Eq. (1), letting

\[ z = \sum_{\xi(v) = 1, \xi^2(w) = 0} |\Gamma(v)| - \sum_{\xi(w) = 0, \xi(w) = 1} |\Gamma(w)|, \quad r = |\Gamma(v_s)|, w = N_2(v_s, 0) \]
we see that Eq. (9) is greater than or equal to \( N_2(v, 0)N_2(v, 1) \). Therefore
\[
E_\xi(U) \geq N_2(v, 0)N_2(v, 1) \left( \sum_{A_1 \subseteq U \setminus \{v\}} \prod_{u \in U \setminus \{v\}} \frac{N_2(u, \xi^{A_1}(u))}{|\Gamma(u)|} \right) 
\]
\[
= N_2(v, 0)N_2(v, 1) \geq 1(d + 1 - 1) = d.
\]
Note that the above inequalities hold, by \( v \in V' \) and Eq. (8). Hence
\[
\sigma_{min}^2(k) \geq \min_{\xi \in \hat{\Xi}} \left\{ \frac{1}{(\xi)} \sum_{U \subseteq \text{Sub}_k(V)} \sum_{U' \cap V' \neq \emptyset} d \right\} = \frac{d}{(\xi)} \min \{ |U \in \text{Sub}_k(V) : U \cap V' \neq 0 | \}
\]
\[
\geq \frac{d}{(\xi)} \left\{ \frac{1}{|U \subseteq \text{Sub}_k(V) : U \cap \{v_1', v_2'\} \neq \emptyset |} \right\} = \frac{d}{(\xi)} \left\{ \left( \frac{n}{k} \right) - \left( \frac{n - 2}{k} \right) \right\}.
\]
This completes the proof. \( \square \)

By Lemma 1 and Proposition 1, we have the following theorem:

**Theorem 3** Let \( n \) and \( d \) be the order and the minimum degree of \( G \), respectively, and let \( E_\xi[T] \) be the expected agreement time of the probabilistic \( k \)-polling game on \( G \) with an initial global state \( \xi \). Then for any \( 1 \leq k \leq n \),
\[
E_\xi[T] \leq \frac{n(n - 1)}{k(2n - k - 1)}d \left( \sum_{v \in V} |\Gamma(v)| \right) \left( \sum_{w \in V \setminus \{v\} = 1} |\Gamma(w)| \right).
\]
\[
(10)
\]

**Proof.** By using Eq. (3) for initial state \( \xi \in \hat{\Xi} \), we obtain the expectation of \( x_t^2 \) for \( t = 0 \) and \( T = \inf \{ t \in \mathbb{N} : x_t^{(0)} \in \{0, 1\} \} \) respectively:
\[
E_\xi[x_0^2] = \left( \sum_{v \in V} |\Gamma(v)| \right) \left( \sum_{w \in V \setminus \{v\} = 1} |\Gamma(w)| \right), \quad E_\xi[\bar{x}_0^2] = \left( \sum_{v \in V \setminus \{v\} = 1} |\Gamma(v)| \right)^2.
\]
\[
(11)
\]

By Lemma 1, it is clear that \((x_t^2 - t\sigma_{min}^2, f_t)\) is a submartingale. We hence apply the optional stopping theorem [5, pp 69, Remark] to \( x_t^2 - t\sigma_{min}^2 \) to have
\[
E_\xi[x_T^2 - T\sigma_{min}^2] \geq E_\xi[x_0^2].
\]

By using Eq. (11) we have
\[
E_\xi[T] \leq \frac{E_\xi[x_T^2] - E_\xi[x_0^2]}{\sigma_{min}^2} = \frac{1}{\sigma_{min}^2} \left( \sum_{v \in V \setminus \{v\} = 0} |\Gamma(v)| \right) \left( \sum_{w \in V \setminus \{w\} = 1} |\Gamma(w)| \right).
\]
\[
(12)
\]
By virtue of Proposition 1, we finally obtain Eq. (10).

As mentioned, Hassin and Peleg [4] adopt for any vertex \( v \), the neighborhood \( \Gamma(v) \) that does not include \( v \) itself. For their setting, we can obtain a similar result:

\[
E_\xi[T] \leq \frac{n(n-1)}{k(2n-k-1)} \left( \sum_{v \in V, \xi_0(v) = 0} d(v) \right) \left( \sum_{v \in V, \xi_1(v) = 1} d(v) \right),
\]

(13)

where \( d(v) \) is the degree of \( v \). Since the order of right hand side of Eq. (13) is \( O(n^4) \) for \( n \)-polling, this bound is weaker than Theorem 2. However for some subclasses of graphs, we can obtain better bounds for \( n \)-polling (for both of their and our settings).

**Corollary 1** (dense) If \( d = \Theta(n) \) then

\[
\max_\xi E_\xi[T] = O(n^3),
\]

where \( d \) is the minimum degree.

**Corollary 1** (non-dense) If \( D = O(1) \) then \( \max_\xi E_\xi[T] = O(n^2) \), where \( D \) is the maximal degree of the graph, that is, \( D = \max_{v \in V} d(v) = \max_{v \in V} |\Gamma(v)| - 1 \).

### 2.2 Complete Graphs

When \( G \) is a complete graph, a global state is characterized by the number 1’s (i.e., vertices with value 1) in it. We therefore use the global state space \( S = \{0, 1, \cdots, n\} \), instead of \( \Xi = \{0, 1\}^V \). By the definition of \( k \)-polling, an \( S \)-valued Markov chain, \( Z_t^{(k)} \), is associated. Let \( p_k(i, j) \) be the transition probability from global state \( i \in S \) to \( j \in S \).

**Lemma 2** \( p_k(i, j) = 1 \), if \( (i, j) \in \{(0,0), (n,n)\} \),

\[
p_k(i, j) = \frac{1}{\binom{k}{l}} \sum_{i=\min\{k,l\}}^{\max\{0,k+i-n\}} \binom{n-i}{k-i-l} \binom{i}{j} \binom{k}{j+i+l} \left( \frac{i}{n} \right)^{j+i+l} \left( 1 - \frac{i}{n} \right)^{k-j+i-l},
\]

(15)

if \( \{ (i, j) : 1 \leq i \leq n-1, \ i - \min\{k,i\} \leq j \leq k + i - \max\{0,k+i-n\} \} \), and \( p_k(i, j) = 0 \) otherwise.

**Proof.** By definition \( p_k(0,0) = p_k(1,1) = 1 \) holds. The distribution concerning \( k \)-polling is hyper-geometrical: Let us randomly select \( k \) vertices from \( V \) constructed
by $i$ vertices with value 1 and $n - i$ vertices with value 0. Then the probability that exactly $l$ vertices in the selected $k$ vertices have value 1 is
\[
\binom{n-i}{k-l} \binom{i}{l} \binom{n}{k}^{-1},
\]
where $\min\{k, i\} \leq l \leq \max\{0, k + i - n\}$. Let $m$ be the number of vertices with value 1 after updating the selected $k$ vertices. The probability of obtaining $m$ vertices with value 1 is calculated by the binomial distribution with parameter $i/n$, that is,
\[
\binom{k}{m} \left( \frac{i}{n} \right)^m \left( 1 - \frac{i}{n} \right)^{k-m}, \quad 0 \leq m \leq k,
\]

because of the completeness of the graph. Moreover the changing number of vertices with value 1 is $m - l$ for the update. On the other hand, assuming the transition from $i$ to $j$, we have that $m = j - i + l$. Since the events of selecting and updating are independent, the transition probability is the sum of the product of Eqs. (16),(17) for possible terms. Hence we have Eq. (15). Clearly the probability is 0, otherwise.

By Lemma 2, the transition probabilities for 1- and $n$-pollings are
\[
\begin{align*}
    p_1(i, j) &= \begin{cases} 
        \left( \frac{i}{n} \right)^2 + \left( 1 - \frac{i}{n} \right)^2, & \text{for } i = j, i \in S, \\
        2\left( 1 - \frac{i}{n} \right)/n, & \text{for } j = i + 1, i - 1, i = 1, \cdots, n - 1, \\
        0, & \text{otherwise},
    \end{cases} \\
    p_n(i, j) &= \binom{n}{j} \left( \frac{i}{n} \right)^j \left( \frac{n-i}{n} \right)^{n-j}, \quad i, j \in S.
\end{align*}
\]

These probabilities are known as Moran’s and Wright-Fisher’s models in population genetics, respectively (E.g., see [1] and [7, Examples 5.1.3 and 5.1.4]), and their expected agreement times are well-known [7, pp 178]:

1-polling \(-n^2 \{ p \ln p + (1 - p) \ln(1 - p) \} (1 + o(1)) = \Theta(n^2)\).

$n$-polling \(-2n \{ p \ln p + (1 - p) \ln(1 - p) \} (1 + o(1)) = \Theta(n)\).

The following theorem that treats $k$-polling is an “interpolation” of them.

**Theorem 4** Let $T^{\text{comp}}$ be the agreement time for $k$-polling on $n$-complete graphs with an initial state satisfying that the proportional ratio of the number of 1’s in the state to $n$ is $p \in (0,1)$. Then
\[
E_{ap}[T^{\text{comp}}] = -2 \left\{ p \ln p + (1 - p) \ln(1 - p) \right\} \frac{n^2(n-1)(1+o(1))}{k(2n-k-1)}. \tag{18}
\]
We omit the proof here, since it is essentially due to [1, pp 75]. The method is based on an approximation of the solution of Eq. (5) for the transition probability in Lemma 2. Some properties are needed for the proof. Let $W_t^{(k)} = Z_t^{(k)} - Z_{t-1}^{(k)}$. Then we also use the following facts: 

\[ E[W_t^{(k)}] = 0 \]

\[ E[(W_t^{(k)})^2 | Z_t^{(k)} = i] = \frac{k(2n - k - 1) i}{n-1} \left( 1 - \frac{i}{n} \right). \] (19)

Note that Eq. (19) corresponds to $\sigma^2_t(k)$ in Proposition 1.

3 Conclusion

In this paper, we gave some upper bounds on the expected agreement time of the probabilistic polling game defined by $k$-polling.

If the graph is uniformly dense or non-dense then the order of expected agreement time seems not to be so large by Corollary 1. Therefore we conjecture that a pair of a graph and an initial global state for it that achieve the worst expected agreement time is given in Fig. 1. The graph there is called barbell graph, which consists of two copies of $n/3$-cliques connected by a path graph of length $n/3$. In the initial global state, the vertices with value 0 (1) are colored white (black). Hence there is only one edge connecting vertices with different values.

![Figure 1: A barbell graph: two copies of $n/3$-cliques are connected by a path graph of length $n/3$. Vertices with value 0 (1) are colored white (black).](image)

References


Toshio Nakata is with the Department of Information Education, Fukuoka University of Education, Akama Munakata, Fukuoka, 811-4192, Japan. E-mail: nakata@crown.fukuoka-edu.ac.jp

Masafumi Yamashita is with Department of Computer Science and Communication Engineering, Kyushu University, Higashiko, Higashi-ku, Fukuoka, 812-8581, Japan. E-mail: mak@csce.kyushu-u.ac.jp
Oblivious Routing on $d$-dimensional Meshes*

A. OSTERLOH
Ilmenau Technical University, Germany

Abstract

We give new upper and lower bounds for oblivious permutation routing on $d$-dimensional meshes ($d > 1$) of side length $n$. We present $O(n^{d/2} \log n)$ algorithms for $d \geq 2$ using queue-size 6. Further we present optimal $O(n^{d/2})$ algorithms using unlimited queue-size. For odd $d$ the presented algorithm is a $d$-bend algorithm. We further show that for odd $d$ an algorithm which restricts for every packet the bends to the minimal number (at most $d - 1$) needs at least $\Omega(n^{(d+1)/2})$ steps.

Keywords

oblivious routing, permutation routing, $d$-dimensional meshes

1 Introduction

One of the most studied parallel models with a fixed interconnection network is the two-dimensional $n \times n$ mesh, in which which $n^2$ processors are connected by a two-dimensional grid of bidirectional communication links. Its natural generalizations are $d$-dimensional $n \times \ldots \times n$ meshes of $N = n^d$ processors, which are connected by a $d$-dimensional grid of bidirectional communication links (for a formal definition see section 2).

The problem of routing packets through a network is fundamental to the study of parallel computation. The canonical routing problem is that of permutation routing. In this case each processor initially and finally contains one packet. Many different approaches to solve the permutation routing problem have been studied (e.g. adaptive, hot-potato, cut-through, wormhole, fault-tolerant, local, etc. [3]). In [16] the concept of oblivious routing was introduced. In this routing strategy the path of each packet is determined only by its source and destination position. It depends not on other packets. This makes oblivious routing simple and attractive and hence it was considered in several publications e.g. [1, 10, 13, 8, 14, 2, 12]. This simplicity has its costs in the running time. In [8] Kaklamanis, Krizanc,
and Tsantilas have shown that every oblivious permutation routing algorithm on a \( N \) node network with maximum node degree \( k \) needs at least \( \Omega(N^{\frac{d}{2}}/k) \) steps and hence an oblivious permutation routing algorithm on the \( d \)-dimensional mesh needs at least \( \Omega(n^{\frac{d}{2}}/d) \) steps. Further Krizanc has shown in [9] that every pure oblivious permutation routing algorithm on a \( d \)-dimensional mesh with constant buffer-size requires at least \( \Omega(n^d) \) steps. Here pure means that if at a particular time there are one or more packets in a queue for an edge then one of them is chosen to use the edge. For a long time it was an open problem whether it is possible to obtain non pure oblivious permutation routing algorithms using constant buffer-size that beat the above \( \Omega(n^d) \) bound on meshes.

Recently Iwama and Miyano have proposed an optimal \( O(n) \) oblivious permutation routing algorithm with buffer-size 2 for the two-dimensional mesh in [5]. In [6] they reduced the leading constants in the running time of their algorithm dramatically at the cost of increasing the constant for their buffer-size. Further Iwama, Kambayashi, and Miyano presented in [7] a technique to obtain an oblivious permutation routing algorithm using constant buffer-size for three-dimensional meshes from an algorithm for two-dimensional meshes. Using a generalization of this technique to higher dimensions and the algorithms presented in [5],[6] we have an \( O(n^{d-1}) \) oblivious permutation routing algorithm using constant buffer-size for \( d > 1 \). Also in [7] Iwama, Kambayashi, and Miyano presented a fast \( 1.16 \cdot n^2 + o(n^{3/2}) \) three bend oblivious routing algorithm for the 3-dimensional mesh using unlimited buffer-size. In a \( k \)-bend algorithm a packet changes its direction at most \( k \) times. If for every packet the number of bends is restricted to minimal then an oblivious routing algorithm needs at least \( \Omega(n^2) \) steps on the 3-dimensional mesh [4].

In this paper we present new upper and lower bounds for oblivious permutation routing on \( d \)-dimensional meshes. We present \( O(n^{d/2} \log n) \) algorithms for \( d \geq 2 \) using queue-size 6. For all \( d \geq 2 \) this is near to the optimal value. The running time is only a factor of \( \log n \) away from the lower bound of \( \Omega(n^{d/2}) \). For \( d \geq 3 \) the presented algorithms are the fastest known algorithms for oblivious permutation routing using \( O(1) \) queue-size on meshes up to now. The previous best known algorithm for \( d \geq 3 \) needs \( O(n^{d-1}) \) time. For the case of unlimited queue-size we present optimal \( O(n^{d/2}) \) algorithms for all \( d \). We further show a lower bound for elementary-path oblivious routing algorithms, for odd \( d > 1 \). In an elementary-path routing algorithm every packet uses the minimal number of bends on its way from its source to its destination. This lower bound result is an extension and a great simplification of a result in [4].

In Section 2 we introduce some basic definitions and results. In Section 3 we introduce our model. In Section 4 we present an \( O(n^{d/2} \log n) \) step oblivious permutation routing algorithm for \( M_{d,n} \) which uses \( O(1) \) buffers-size. In Section 5 we give optimal oblivious routing algorithms for \( M_{d,n} \) with unlimited buffer-size and show a lower bound for elementary-path oblivious routing algorithms.
2 Basic Definitions and Results

For \( x \in \mathbb{N} \) let \( [x] \) be the set \( \{0, \ldots, x - 1\} \subset \mathbb{N} \).

**Definition 1** A \( d \)-dimensional mesh \( M_{d,n} \) of side length \( n \) is an undirected graph with node set \( P_{d,n} = \{(p_0, \ldots, p_{d-1}) \in [n]^d \} \) and edge set \( E_{d,n} = \{((p_0, \ldots, p_{d-1}), (q_0, \ldots, q_{d-1})) \in P_{d,n} \times P_{d,n} | \sum_{i=0}^{d} |p_i - q_i| = 1 \} \).

The set \( P_{d,n} \) represents the processors and the set \( E_{d,n} \) represents the bidirectional communication links of the network.

In this paper we consider oblivious permutation routing of \( M_{d,n} \). In the permutation routing problem every processor on \( M_{d,n} \) initially has one packet. We call this processor the source of the packet. Additionally each packet has a destination such that the set of all destinations is equal to \( P_{d,n} \). The problem is to send the packets along a path in \( M_{d,n} \) from their source to their destination. In oblivious routing the path a packet uses is determined only by its source-destination pair. The path is completely independent of the path used by the other packets.

For our routing algorithm we divide the mesh \( M_{d,n} \) in substructures (subgraphs) called blocks. We use two kinds of substructures resulting in two kinds of blocks. The source blocks and the destination blocks. A packet has source block \( S \) or belongs to source block \( S \) if its source is in block \( S \). A packet has destination block \( D \) if its destination is in block \( D \). We omit the formal definition of blocks here to avoid an overflow of notation. The exact definition of the blocks will be given in Section 4.

Now we define the so called bit-reversal permutation (e.g. see [11]). In our algorithm we use this permutation to distribute packets within the source blocks.

**Definition 2** Let \( l \in \mathbb{N} \). For \( b \in [2^l] \), let \( b_{l-1} \ldots b_0 \in \{0,1\}^l \) be the binary representation of \( b \) of length \( l \), i.e. \( b = \sum_{i=0}^{l} b_i \cdot 2^i \). The bit-reversal permutation \( brp_l : [2^l] \rightarrow [2^l] \) maps \( b \) to \( brp_l(b) = \sum_{i=0}^{l} b_{l-i} \cdot 2^{2l-i-1} \).

We use \( brp_l \) to distribute packets with the same destination block within its source block. To achieve this we first sort the packets such that packets with the same destination block are placed in neighbouring processors and then we apply \( brp \). If we apply this technique we can give a lower bound for the distance of packets with the same destination block. The following Lemma holds (see [5] for a similar result):

**Lemma 1** Let \( l \in \mathbb{N} \), \( t \in [l + 1] \), \( x \in [2^{l-t}] \), and \( i, j \in [2^{l+1}] \), \( i \neq j \). Then we have \( |brp_{l+1}(x \cdot 2^{l+1} + i) - brp_{l+1}(x \cdot 2^{l+1} + j)| \geq 2^{l-t} \).

**Proof.** \( |brp_{l+1}(x \cdot 2^{l+1} + i) - brp_{l+1}(x \cdot 2^{l+1} + j)| = |i - j| \cdot 2^{l-t} \geq 2^{l-t} \). \( \square \)

In our algorithm we do not really sort the packets. We calculate for every packet \( p \) a number \( v_p \) by counting and then we transport the packet to \( brp(v_p) \).
3 Our Model

In our model each processor has $2d$ input and $2d$ output queues. One input queue and one output queue for each of its $2d$ links. Each queue can hold up to $Q \in O(1)$ packets at the same time. A packet consists of four parts (see Figure 1), the source address, the destination address, the message, and the part for additional information.

Source and destination address need $O(\log n)$ bits. We restrict the additional information to $O(\log n)$ bits and assume that the messages have uniform length of $m$ bits. So every processor must be able to store at least $4dQ(m + \Theta(\log n))$ bits. Hence we can assume that every processor can use $O(\log n)$ bits for computations.

Our time model is the same as in [5]. One time-step consists of the following two computation steps: (i) Assume that there are $l$ packets in an output queue $O$ of a processor $P$. Then $P$ select at most $Q$ packets from its input queues and move them to $O$. (ii) Let $P$ and $P'$ be neighboring processors, let $I_P$ be the input queue for $P$ at $P'$ and $O_P$ be the output queue for $P'$ at $P$. If $I_P$ has a free space then $P$ selects at most one packet from $O_P$ and send it to $P'$. Note that the decisions in (i) and (ii) depend on the used algorithm. In the oblivious setting the link which a packet uses depends not on the other packets in the queues. The algorithm is allowed to use any information (e.g. additional information) to decide the time step when a packet is sent over its link. Note that in contrast to the model used in [9] in our model no buffer overflow is possible. In our model blocking caused by processors with full buffers can occur.

4 Routing with Constant Buffer Size

4.1 Case $d = 3$.

In the following we assume $n = 2^l$ for an even integer $l \geq 2$. The case $l = 0$ is trivial and for $l = 1$ the mesh is a $d$-dimensional hypercube and the problem is solved in [8] in $O(\frac{\sqrt{N}}{n})$ time ($N = 2^d$). For the case $d = 3$ the algorithm in [8] needs at most 3 buffers. The case that $n$ is not an even power of 2 will be discussed at the end of this section.

We divide the 3-dimensional mesh $M_{3,n}$ in blocks of $n\sqrt{n}$ processors. We use two kinds of blocks. The source blocks and destination blocks. There are $n\sqrt{n}$ blocks of each kind.
Definition 3 Let $i \in [n], j \in \lceil \sqrt{n} \rceil$. The source block $S_{i,j}$ consists of processors $\{(i, y, z) \in P_{3,n} | z \in [n], j\sqrt{n} \leq y < (j+1)\sqrt{n}\}$. The destination block $D_{i,j}$ consists of processors $\{(x, y, i) \in P_{3,n} | x \in [n], j\sqrt{n} \leq y < (j+1)\sqrt{n}\}$. We denote the set of all source blocks by $S$.

In Figures 2,3 we show the source and destination blocks of a $16 \times 16 \times 16$ mesh.

Now we define entry places and exit places. For this purpose we define mappings $\text{entry}_{B}$ and $\text{exit}$. An entry place $\text{entry}_{B}(B')$ in destination block $B = D_{i,j}$ ($i \in [n], j \in \lceil \sqrt{n} \rceil$) for packets from source block $B'$ is the processor of destination block $B$ to which $B'$ sends packets with destination in $B$. At this processor such packets 'enter' their destination block. Here enter a block for a packet means that it tries to begin to travel on a special (Hamiltonian) path within this block. An exit place $\text{exit}(B)$ is the processor of source block $B$ where packets with source block $B$ 'leave' their block. Here leaving its block for a packet means that it stops using a special (Hamiltonian) path in its source block and begins to travel to its destination block. The exit place of a block $B$ can be chosen rather freely.

Definition 4 For $i \in [n], j \in \lceil \sqrt{n} \rceil$:

\[
\begin{align*}
\text{entry}_{D_{i,j}} : & \quad S \quad \rightarrow \quad D_{i,j} \\
& \quad \quad S_{k,l} \quad \rightarrow \quad (k, l + j\sqrt{n}, i) \\

\text{exit} : & \quad S \quad \rightarrow \quad P_{3,n} \\
& \quad \quad S_{k,l} \quad \rightarrow \quad (k, l + 1\sqrt{n}, l\sqrt{n})
\end{align*}
\]

For $\text{entry}_{D_{i,j}}$ we also write $\text{entry}_{i,j}$. 

Figure 2: Source blocks of a $16 \times 16 \times 16$ mesh.
Note that for all $i \in [n]$, $j \in [\sqrt{n}]$ the mapping $entry_{i,j}$ is bijective. We now give for every source block $B$ and for every $i \in [n], j \in [\sqrt{n}]$ a path $path(B, i, j)$ from its exit place to the entry place $entry_{i,j}(B)$. The path is given as the unique path from exit($B$) to entry($B$) in a tree $tree(B)$ with root exit($B$). In the following we write $\ast$ if we want to denote all elements of the set $[n]$.

**Definition 5** For $i \in [n], j \in [\sqrt{n}]$ we define $tree(S_{i,j}) = (V_{i,j}, E_{i,j})$, where $V_{i,j} = (i, \ast, j, \sqrt{n}) \cup \bigcup_{k \in [\sqrt{n}]} (i, k, \sqrt{n} + j, \ast)$ and $E_{i,j} = (V_{i,j} \times V_{i,j}) \cap E_{n,n}$. Here $(i, \ast, j, \sqrt{n}) = \{(i, y, j, \sqrt{n}) \in P_{n,n} | y \in [n]\}$ (analogously we define $(i, k, \sqrt{n} + j, \ast)$). The root of $tree(S_{i,j})$ is exit($S_{i,j}$).

Figure 4 shows the $(i, \ast, \ast)$ plane of a $16 \times 16 \times 16$ mesh. The exit places of the four blocks in this plane are shown as black filled circles. The tree of source block $S_{i,2}$ is shown with dashed lines and the entry place of destination block $D_{2,1}$ for packets from block $S_{i,2}$ is shown as a white filled circle.

Because $n$ is even there exists an Hamiltonian cycle in every block. In a block we number the processors opposite to the cycle from 0 to $n\sqrt{n} - 1$ beginning with exit($B$) (see Figure 5) and give every processor two places. The processor with number $i$ gets the places $2i$ and $2i + 1$.

We give every destination block $D$ an unique number $b_D$ in $[n\sqrt{n}]$. Now let $S$ be an arbitrary source block. We define an ordering of the destination blocks in $S$. We write $\#_{S,D}$ for the number of packets in $S$ with destination in $D$. For two destination blocks $D_1$ and $D_2$ we define $D_1 \leq_S D_2 :\iff (\#_{S,D_1} < \#_{S,D_2}) \lor ((\#_{S,D_1} = \#_{S,D_2}) \land (b_{D_1} < b_{D_2})).$

We now define an ordering on the packets in $S$. For a packet $p$ let $D_p$ be its destination block and $h_p$ be the number of its source processor on the Hamiltonian cycle in $S$. Then for two packets $p, p'$ in $S$ we define $p \preceq_S p' :\iff (D_p \leq_S D_{p'}) \lor
Figure 4: The (i,*,*) plane of a $16 \times 16 \times 16$ mesh.

Figure 5: An Hamilton cycle in a block of a $16 \times 16 \times 16$ mesh.
((D_p = D_{\hat{p}}) \land (h_p < h_{\hat{p}}))). For packets \( p, p' \) in \( S \) with the same destination block \( \preceq_S \) implies an ordering by \( h_p < h_{\hat{p}} \). We denote this ordering by \( \sqsubseteq_S \).

Now we can present the routing algorithm. It works as follows: In step 1 we calculate some ranks. This is done by counting. In step 2 we distribute packets with the same destination block in the Hamiltonian cycle in their source blocks. This is done by using the bit-reversal permutation. In step 3 we transport the packets to their destination. To avoid congestion at the entry places this is done in phases. In a phase only packets \( p \) in a source block \( S \) with approximately the same number \( d_p = \#_{S,D_p} \) have the permission to move. We call such packets active. To give packets more time to enter their destination block at the entry place before the next packet with the same destination block arrives we enlarge the distance between packets by a factor \( space \). The size of \( space \) is calculated in the analysis of algorithm 3D.

Algorithm 3D:

1: For all \( i \in [n], j \in [\sqrt{n}] \), for all packets \( p \) in block \( S_{i,j} \): Calculate the rank \( r_p \) of \( p \) with respect to \( \preceq_{S_{i,j}} \), the rank \( g_p \) of \( p \) with respect to \( \sqsubseteq_{S_{i,j}} \), and \( d_p = \#_{S_{i,j}, D_p} \). Let \( t_p \) be such that \( 2^t \cdot h_p < d_p < 2^{t+1} \). Calculate \( v_p := r_p + 2^t - ((r_p - g_p) \mod 2^{t+1}) \).

2: For all \( i \in [n], j \in [\sqrt{n}] \), for all packets \( p \) in block \( S_{i,j} \): If \( 0 \leq \lfloor brp_{3l/2+1}(v_p) / 2 \rfloor \leq h_p \) (remember \( h_p \) is the number of the source processor of \( p \) on the Hamiltonian cycle in \( S_{i,j} \)): Transport \( p \) along the Hamiltonian cycle such that it makes a full cycle. Then transport the packet to place \( brp_{3l/2+1}(v_p) \). If \( h_p < \lfloor brp_{3l/2+1}(v_p) / 2 \rfloor < n/\sqrt{n} \): Transport \( p \) along the Hamiltonian cycle to place \( brp_{3l/2+1}(v_p) \).

3: This step consists of \( \lceil \frac{3}{2} l \rceil \) phases. In phase \( t \in [\lceil \frac{3}{2} l \rceil] \) only packets \( p \) with \( 2^t \cdot d_p < 2^{t+1} \) are active. For all \( i \in [n], j \in [\sqrt{n}] \), for all active packets \( p \) in block \( S_{i,j} \): Let \( D_{l(p),l(p)} \) be the destination block of \( p \). Transport \( p \) along the Hamiltonian cycle to \( exit(S_{i,j}) \) and then along \( path(S_{i,j}, k(p), l(p)) \) to \( entry(S_{i,j}, k(p), l(p)) \). In its destination block transport \( p \) along the Hamiltonian cycle to its destination processor. In the case \( t = 0 \) let \( T_i \) be the time step when step 2 is finished. If \( t > 0 \) let \( T_i \) be the time step when phase \( t - 1 \) is finished. For an active packet on place \( m \in [2n/\sqrt{n}] \) the transport begins in time step \( T_i + 1 + 2 \cdot space \cdot m \).

The ranks \( r_p, g_p \) and the number \( d_p \) in step 1 of algorithm 3D can be calculated in the following way with \( O(\log n) \) bits on every processor and \( O(\log n) \) bits in the part for additional information of every packet. In the beginning every processor remembers the destination block of its packet. Then packets are moved around the Hamiltonian cycle one time. On this cycle every packet \( p \) can calculate \( g_p \) and \( d_p \) simply by counting. After the first cycle every processor remembers \( g_p \) and \( d_p \) of
its packet. In a second cycle every packet can calculate \( r_p \) by counting using the information calculated in the first cycle.

Note that \( g_p \in [d_p] \). Let \( p \) be a packet in \( S_{i,j} \) with \( g_p = 0 \). Then \( p \) has the lowest rank (with respect to \( S_{i,j} \)) of the packets in \( S_{i,j} \) with destination block \( D_p \). Further for every packet \( p' \) in \( S_{i,j} \) with \( D_{p'} = D_p \) we have \( r_{p'} = r_p + g_{p'} \) and \( r_p \leq r_{p'} < r_p + d_p \). So \( v_p \mod 2^p = 0, v_{p'} \mod 2^p = g_{p'} \) and \( v_p \in [2n\sqrt{n}] \). Note further that every packet gets a unique \( v_p \). For two packets \( p \) and \( p' \) in \( S_{i,j} \) with \( D_p = D_{p'} \) we get with Lemma 1 that \( |brp3j/2+1(v_p) - brp3j/2+1(v_{p'})| \geq 2^{3/2-t_{i,j}+1} \geq \frac{2n\sqrt{n}}{d_p} \geq \frac{n\sqrt{n}}{d_p} \).

In step 2 the additional cycle for some packets is necessary because without this the path of packets would depend on other packets. With this extra cycle every packet cycles three times (two cycles in step 1 and one cycle in step 2) and then goes to the exit place. Note that it is no problem if there are processors with no packet initially. The only consequence is that in this case there are also processors with no packets after step 2. So algorithm 3D can also be used for problems where every processor is source and destination of at most one packet (so called 1-1 routing).

Obviously the steps 1, and 2 can be performed in \( O(n\sqrt{n}) \) steps. Now we have to consider the step 3.

First note that the Hamiltonian cycles in the source and destination blocks are edge disjoint. Also tree \( S_{i,j} \) and tree \( S_{i',j'} \) are edge disjoint if \( (i,j) \neq (i',j') \). But the trees are not edge disjoint with the cycles. To avoid edge congestion in step 3 we interleave the transport of the packets in the following way. In time step 1 of a phase the packets on the Hamiltonian cycles in the source and destination blocks are moved, in time step 2 of a phase the packets in the trees are moved, in time step 3 of a phase again the packets on the Hamiltonian cycles in the source and destination blocks are moved, in time step 4 the packets in the trees are moved etc.

Now we come to the critical part of the analysis of algorithm 3D. We have to show that the packets can enter the Hamiltonian cycle of their destination block at the entry place before the next packet for the same block arrive. We give priority to packets on the Hamiltonian cycle, i.e. if a packet is on the Hamiltonian cycle in its destination block it moves to its destination processor without any further delay. Note that in phase \( t \) in each destination block at most \( \frac{n\sqrt{n}}{d_p} \) entry places can receive packets. We call entry places which receive packets in phase \( t \) active. The time distance between two packets arriving at the same active entry place is at least \( 2 \cdot \text{space} \cdot \frac{n\sqrt{n}}{d_p} \) due to Lemma 1 and the fact that the first transport of an active packet on place \( m \) in phase \( t \) is in time step \( T_i + 1 + 2 \cdot \text{space} \cdot m \).

**Lemma 2** For \( \text{space} > 4 \) the following holds: If in phase \( t \) at most two packets are inserted at every active entry place into the Hamiltonian cycle of a destination block in \( 2 \cdot \text{space} \cdot \frac{n\sqrt{n}}{d_p} \) time then at least two packets can be inserted at every active
entry place into the Hamiltonian cycle of a destination block in \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time.

**Proof.** We consider a worst case scenario. Let \(x\) be the active entry place with the highest number in the cycle. Assume that all other active entry places send their packets to a processor such that these packets have to pass \(x\) and can possibly block a packet that wants to enter the cycle at \(x\).

We now count the number of packets that can block a packet at \(x\). In \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time steps every active entry place can send at most two packets. These packets are not delayed on their way to \(x\) because packets on the cycle have higher priority than packets that want to enter the cycle. So in \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time steps at most two packets from each active entry place arrive at \(x\). In phase \(t\) there are at most \(\frac{n}{2^{l+1}}\) active entry places in a destination block. Hence the total number of packets that want to pass \(x\) in \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time steps is \(2 \cdot \left(\frac{n}{2^{l+1}} - 1\right) = \frac{n}{2^{l+1}} - 2\).

For \(\text{space} > 4\) there are at least two time slots in \(\text{space} \cdot \frac{n}{2^{l+1}}\) time at the entry place \(x\) where no packets want to pass \(x\). At these time slots the packets can be inserted in the cycle.

Due to the above Lemma a packet arriving at an active entry place at some time step \(t_0\) can enter the cycle before time step \(t_0 + \text{space} \cdot \frac{n}{2^{l+1}}\). So there is no more than one buffer needed at the entry places. Note that it is possible that an active entry place inserts two packets in the cycle in \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time. This can happen if a packet is blocked at the entry place before it is inserted. If the next packet arrives \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time steps after the first one has arrived at the entry place and can enter the cycle without blocking then the entry place has inserted two packets in the cycle in \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time. Obviously it is impossible that an entry place insert more than two packets in the cycle in \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) time (Lemma 1, definition of step 3).

Now we can calculate how long phase \(t\) take. The last packet begins its movement at most \(4 \cdot \text{space} \cdot n\sqrt{n}\) time steps after the beginning of the phase. It has to travel a distance of at most \(2n + 2n\sqrt{n}\) delayed by a factor 2 due to the interleaving, and it has to wait at most \(2 \cdot \text{space} \cdot \frac{n}{2^{l+1}}\) steps at the entry place. Hence phase \(t\) take at most \(O(n\sqrt{n})\) steps. So algorithm 3D takes at most \(O(n\sqrt{n}\log n)\) steps.

Now we have to consider the case that \(n\) is not an even power of 2. Let \(2^{l} < n < 2^{l+2}\) for an even integer \(l > 0\). Then we cover the mesh \(M_{3,n}\) with 64 sub-meshes of size \(2^l \times 2^l \times 2^l\). We denote the sub-meshes with \(M_i\) for \(1 \leq i \leq 64\). We have 64 possible cases where the source and destination of a packet can be. In the case \((i, j)\) the source of a packet is in mesh \(M_i\) and the destination in \(M_j\). The case \((i, j)\) can be solved as follows. The packets from mesh \(M_i\) with destination in \(M_j\) are moved to \(M_j\) without changing their relative positions in \(M_i\). Then an 1-1 routing problem in \(M_j\) is solved. The movement from \(M_i\) to \(M_j\) can be done in \(O(n)\) and the 1-1 routing problem can be solved in \(O(n\sqrt{n}\log n)\) time using
algorithm $3D$. Hence the whole problem can be solved in $O(n\sqrt{n}\log n)$ time.

Finally we have to consider the queue-size. At every processor we need three buffers for the interleaving and $brp$, one buffer for the non active packets, and one buffer for the packets at the entry places. One additional buffer is needed if $n$ is not an even power of two. So the queue-size can be bounded by 6.

**Theorem 1** There exists an oblivious permutation routing algorithm on the mesh $M_{3,n}$ which runs in $O(n\sqrt{n}\log n)$ steps and uses queue-size $Q = 6$.

### 4.2 Case $d > 3$ and Case $d = 2$

Our algorithm can also be used for two-dimensional meshes. Assume that $n$ is even. In this case two neighboring rows build a source block and two neighboring columns build a destination block. So in each block an Hamiltonian cycle exists. Here the Hamiltonian cycles must be defined carefully to avoid that the cycles in the source and destination blocks use a link in the same direction. This can be easily done and we omit the details.

Let $S$ be a source block and $D$ be a destination block. Then the cut of $S$ and $D$ consists of four processors. One of these four processors can be chosen to be the entry place in destination block $D$ for packets from source block $S$. Every processor of source block $S$ can be chosen to be the exit place for $S$. So for $d = 2$ no trees are needed to connect entry and exit places. To reach their entry places the packets are moved on the Hamiltonian cycles in the source blocks.

This idea can be used for all $d$-dimensional meshes if $d$ is even, e.g. for $d = 4$ the blocks are two-dimensional planes with $n^2$ processors. There are $n^2$ source blocks $S_{i,j} = (i, j, *, *)$ and $n^2$ destination blocks $D_{k,l} = (*, *, k, l)$ and the cut of a source block $S_{i,j}$ and a destination block $D_{k,l}$ consists of exactly one processor $(i, j, k, l)$. This is the entry place in block $D_{k,l}$ for packets from block $S_{i,j}$. With an analogous analysis as for the case $d = 3$ we can show the following:

**Theorem 2** Let $d > 0$ be an even integer. There exists an oblivious permutation routing algorithm on the mesh $M_{d,n}$ which runs in $O(n^{d/2} \log n)$ steps and uses queue-size $Q = 6$.

For odd $d > 3$ we define source, destination blocks, entry and exit places as follows. Let $m := \frac{d-1}{2}$, $q = (q_1, \ldots, q_{m-1}) \in [n]^{m-1}$. For $i, k \in [n]^m, j, l \in [\sqrt{n}]$ we have $S_{i,j} := \{(i, y, z) \in P_{d,n} \mid z \in [n]^m, j\sqrt{n} \leq y < (j + 1)\sqrt{n}\}, D_{i,j} := \{(x, y, i) \in P_{d,n} \mid x \in [n]^m, j\sqrt{n} \leq y < (j + 1)\sqrt{n}\}, exit(S_{i,j}) = (i, j + j\sqrt{n}, j\sqrt{n}, q)$ and $entry_{i,j}$ ($S_{i,j}$) = $(k, l + j\sqrt{n}, i)$. In this case we have to define paths to connect exit places and entry places. We now give the path from source block $S_{i,j}$ to destination block $D_{k,l}$ with $k = (k_1, \ldots, k_m) \in [n]^m$: 


Theorem 3 Let $d > 1$ be an odd integer. There is an oblivious permutation routing algorithm on the mesh $M_{d,n}$ which runs in $O(n^{d/2} \log n)$ steps and uses queue-size $Q = 6$.

5 Optimal Routing with Unlimited Buffers

In the basic greedy routing algorithm using $\pi \in S_d$ (BGA for short) for $d$-dimensional meshes the packets correct dimensions in the order given by $\pi$. BGA is an oblivious routing algorithm. Every packet is routed along a shortest path and for every packet the number of bends is minimal. It is easy to see that BGA solve the partial permutation problem in $2n^{d/2} + o(n^{d/2})$ time for even $d$ and in $\Theta(n^{(d+1)/2})$ time for odd $d$ if unlimited buffer capacity is available. For even $d$ this is optimal. We now give an optimal algorithm for odd $d$. In [7] an optimal algorithm for the case $d = 3$ is presented. We extend their algorithm to all odd $d > 1$. In the following we set $k = (d - 1)/2$ and assume $n^{1/2}$ is an integer. We use the furthest destination first priority scheme.

Algorithm odd:

1: If $k > 1$: Correct dimensions 0 to $k-2$ in a greedy fashion.

2: Let $p$ be a packet on processor $(p_0, \ldots, p_{d-1})$ at the beginning of this step. Let $(p_0, \ldots, p_{k-2}, q_{k-1}, \ldots, q_{d-1})$ be the destination of $p$. Move $p$ to processor $(p_0, \ldots, p_{k-2}, [q_{k-1}/n^{1/2}] \cdot n^{1/2} + [p_k/n^{1/2}], p_k, \ldots, p_{d-1})$. 

\[
\text{path}(S_{i,j,k,l}) \equiv \begin{cases} 
\text{exit}(S_{i,j}) & \text{if } l = 0 \\
(i, j + l\sqrt{n}, j\sqrt{n}, q_l) & \text{if } l = 1 \\
(i, j + l\sqrt{n}, k_1, q_{l-1}) & \text{if } l = 2 \\
(i, j + l\sqrt{n}, k_2, q_{l-1}) & \text{if } l = 3 \\
& \vdots \\
(i, j + l\sqrt{n}, k_{m-1}, q_{m-1}) & \text{if } l = m \\
\text{entry}_{k,l}(S_{i,j}) & \text{if } l = m+1
\end{cases}
\]

Here $x \rightarrow y$ for $x,y \in P_{d,n}$ means a shortest path from $x$ to $y$. We have $\text{tree}(S_{i,j}) = \bigcup_{k \in [\sqrt{n}, \sqrt{n}]^n} \text{path}(S_{i,j,k,l})$. For $d > 3$ the trees are constructed analogously to the three-dimensional case, such that there is no edge congestion between $\text{tree}(S_{i,j})$ and $\text{tree}(S_{\ell,f})$ if $(i,j) \neq (\ell,f)$. As in the case $d = 3$ the root of $\text{tree}(S_{i,j})$ is the processor $\text{exit}(S_{i,j})$ and the path from source block $S_{i,j}$ to destination block $D_{i,j}$ is given as the unique path in $\text{tree}(S_{i,j})$ from the root to $\text{entry}_{k,l}(S_{i,j})$. With an analogous analysis as for the case $d = 3$ we can show the following:
3: Correct dimension $k$ in a greedy fashion.

4: Correct dimension $k - 1$ in a greedy fashion.

5: Correct dimensions $k + 1$ to $d - 1$ in a greedy fashion.

We now analyze the running time of algorithm odd using the routing lemma of [15] several times. Step 1 can be done in $O(n^{d-1})$ time. After step 1 there are at most $n^{k-1}$ packets on each processor. So step 2 can be done in $O(n^k)$ time. After step 2 there are at most $O(n^k)$ packets on every processor. In every row of dimension $k$ there are at most $n^{1/2}$ processors with packets (to get a row of dimension $k$ fix all dimensions $\neq k$ and let the $k$-th dimension vary from 0 to $n - 1$). Furthermore in any row of dimension $k$ the processors with packets are located in a sub row of length $n^{1/2}$. Hence step 3 can be done in $n^{k+(1/2)} + o(n^{k+(1/2)})$ time. After step 3 every packet has corrected dimensions $0, \ldots, k - 2, k$ exactly and dimension $k - 1$ within a range of $n^{1/2}$. Additionally in step 4 each processor is destination of at most $n^k$ packets. So step 4 can be performed in $n^{k+(1/2)} + o(n^{k+(1/2)})$ time. Step 5 can be done in $O(n^k)$ time.

**Theorem 4** There is an oblivious permutation routing algorithm on $d$-dimensional meshes ($d > 1$) of side length $n$ which runs in $2n^{d/2} + o(n^{d/2})$ steps.

In [4] oblivious elementary-path permutation routing algorithms (EPA) are considered. In EPA the path of each packet must be a shortest path and the number of bends must be minimal. The difference to BGA is that the dimensions can be corrected in any of $d!$ orders. Because we deal with oblivious routing the order chosen for a packet must be determined only by its source and destination. It was shown in [4] that every oblivious elementary-path permutation routing algorithm on $M_{d,n}$ needs $\Omega(n^2)$ steps. We now show that any EPA on a $d$-dimensional mesh for odd $d > 1$ needs $\Omega(n^{(d+1)/2})$ steps. Hence for all $d$ algorithm BGA is an optimal oblivious elementary-path permutation routing algorithm.

In the following $d > 1$ is an odd integer. For $t \in [n]^d$, $\pi \in S_d$ we define $S(t, \pi) = \{ s \in [n]^d | s$ initially on processor $s$ with destination $t$ uses $\pi \}$. For every $t \in [n]^d$ we have $n^d$ sources. There are $d!$ permutations $\pi \in S_d$. Hence for all $t \in [n]^d$ there exist $\pi \in S_d : |S(t, \pi)| \geq \frac{n^d}{d!}$. There are $n^d$ destinations $t \in [n]^d$ and $d!$ permutations, so we have $\exists T \subseteq [n]^d, |T| \geq \frac{n^d}{d!}$ $\exists \pi \in S_d \forall t \in T : |S(t, \pi)| \geq \frac{n^d}{d!}$. Without loss of generality we can assume that $\pi = \text{id}$. If $\pi \neq \text{id}$ then we simply have to permutate the dimensions according to $\pi$ in the rest of our analysis. For $x \in [n]^k$ we define $M_x = \{ (y_1, \ldots, y_{k+1}, x) \in P_{d,n} | y_1, \ldots, y_{k+1} \in [n] \}$ and $\overline{M}_x = \{ (x, y_1, \ldots, y_{k+1}) | y_1, \ldots, y_{k+1} \in [n] \}$. There must be an $x \in [n]^k$ such that $|T \cap M_x| \geq \frac{n^{k+1}}{d!}$. We set $T_x = T \cap M_x$. For all $t \in T_x$ we have $|S(t, \text{id})| \geq \frac{n^k}{d!}$. So for every $t \in T_x$ there are $\Theta(n^k)$ elements $y \in [n]^k$ such that $|S(t, \text{id}) \cap \overline{M}_y| = \Theta(n^{k+1})$. We define $T_{x,y} = \{ t \in T_x | |S(t, \text{id}) \cap \overline{M}_y| = \Theta(n^{k+1}) \}$. There are $\Theta(n^{k+1})$ elements in $T_x$ and so there
exists a \( y \in [n]^k \) such that \( |T_{x,y}| = \Theta(n^{k+1}) \). Using \( T_{x,y} \) and \( M_x \), it is easy to see that a permutation \( perm \) exists which fulfills the following two properties:

1. After correcting \( k \) dimensions of \( perm \) there are \( \Theta(n^{k+1}) \) packets on the line \( L = \{(x, i, y) \in P_{d,n} \mid i \in [n]\} \).

2. While correcting the dimension \( k + 1 \) of \( perm \) this packets have to travel a distance of \( \Theta(n^{k+2}) \).

There are at most \( n \) links in \( L \) and at most one packet can go over a link in one time step. So correcting the dimension \( k + 1 \) of \( perm \) will need at least \( \Theta(n^{k+1}) \) steps.

**Theorem 5** For odd \( d \) an oblivious and elementary-path permutation routing algorithm on the mesh \( M_{d,n} \) needs \( \Omega(n^d) \) steps.

**Corollary 1** For all \( d \) the algorithm \( BGA_{id} \) is an optimal oblivious elementary-path permutation routing algorithm on the mesh \( M_{d,n} \).

**References**


Andre Osterloh is with Ilmenau Technical University, Faculty of Informatics and Automation, Institute of Theoretical and Technical Informatics, Automata and Formal Languages, PF 10 05 65, 98684 Ilmenau, Germany. E-mail: osterloh@theoinf.tu-ilmenau.de
Broadcasting in All-Output-Port Cube-Connected Cycles with Distance-Insensitive Routing

PETR SALINGER
Czech Technical University in Prague, Czech Republic

PAVEL TVRDÍK
Czech Technical University in Prague, Czech Republic

Abstract

In this paper, we consider the problem of one-to-all broadcast (OAB) in the $n$-dimensional cube-connected cycles, CCC$_n$, under the following conditions. The routers use distance insensitive switching, e.g., wormhole. A OAB proceeds in rounds, where a round consists of message passing between pairs of nodes using disjoint paths. The routers also provide all-output-port functionality, which allows the routers to send a packet via all output ports simultaneously. In CCC$_n$, this assumption implies that the lower bound on the number rounds of OAB is $\log_4(|V(\text{CCC}_n)|)$. The main result of this paper is an algorithm which completes a OAB in CCC$_n$ in the number of rounds equal to the lower bound or 1 extra round is needed for all $n < 123$. The algorithm is depth-contention-free and therefore, it works correctly even if nodes participating in the broadcast run asynchronously. We also show how to make the algorithm deadlock-free.

Keywords
cube-connected-cycles, broadcast, distance-insensitive routing, wormhole

1 Introduction

One-to-all broadcast (OAB) belongs to the most important collective communication operations. One sender disseminates the same information to all other nodes. OAB appears in many parallel algorithms in linear algebra, neural networks, optimizations problems, and so on, and is an essential part of communication libraries.

Algorithms for efficient OAB depend strongly on the hardware capabilities of the communication subsystem, mainly on the switching techniques and channel

*This research has been supported by MSMT under research program #J04/98:212300014, by ČVUT under grant #300009203, and by FRVŠ under grant #580/2000.
utilization. Most interconnection networks of massively parallel computers are
based on some kind of distance insensitive packet switching, such as wormhole or
virtual cut-through. These networks are referred to as WH networks in the further
text. A common assumption then is that a collective communication operation
consists of rounds and one round consists of pairs of processors communicating
along edge-disjoint paths simultaneously.

If a collision between paths from different sources in successive rounds can
appear, the rounds must be synchronized, for example, by barriers. Hence, a new
round can start only if all the communication paths from the previous rounds are
released. However, barriers may introduce substantial overhead. If no collision
between paths can appear even if the communication paths are established asyn-
chronously, then the algorithm is said to be depth-contention-free. This property
leads to both implementation simplicity and efficiency. Each node can start a new
round independently of the other nodes, as soon as it receives the packet or fin-
ishes its previous round.

Another common feature of routers is the all-output-port capability, or more
generally, the multicast capability. A packet received by a router in one round
is stored in its processor’s memory and it can be injected into the network via
all its output external channels in the following rounds. This capability implies a
very strong lower bound on the number of rounds for OAB. If $G$ is an $N$-node $d$-
regular network, then the lower bound is $\rho_{OAB}(G) = \lceil \log_{d+1} N \rceil$. It is achieved if
each node, once informed, is able to inform $d$ uninformed nodes in all remaining
rounds. Then $n_t$, the number of informed nodes after round $t$, is $n_t = n_{t-1} + d \cdot n_{t-1} = (d + 1)n_{t-1} = (d + 1)^t$. Optimal OAB algorithms, which match this lower
bound, have been designed for several regular topologies, see Section 3.

In this paper, we propose a nearly optimal OAB algorithm in all-output-
port WH $n$-dimensional cube-connected cycles. For network instances of prac-
tical sizes, our algorithm needs at most one extra round compared to the trivial
lower bound. It is also noteworthy that our algorithm uses dimension-ordered
routing and is depth-contention-free. Since cube-connected cycles are spanning
subgraphs of wrapped butterflies, the same algorithm provides an efficient OAB
in butterflies.

2 Cube-connected cycles

Let $B$ denote the binary alphabet $\{0,1\}$ and $B^n$ the set of all binary strings
of $n$ bits. The inversion of bit $i$ in string $c \in B^n$ is written as $\text{neg}_i(c)$. The $n$-
dimensional cube-connected cycles, denoted by $\text{CCC}_n$, belong to sparse hypercu-
bic networks, a network family containing also butterflies. $\text{CCC}_n$ are derived from
the $n$-dimensional binary hypercube by replacing each hypercube node with an
$n$-node cycle, so that each cycle node is incident with a single hypercube edge.
Hence, $\text{CCC}_n$ have $n2^n$ nodes and $3n2^{n-1}$ edges. Nodes are labeled with pairs
Figure 1: Cube-connected cycles $CCC_3$.

$(i, c)$, where $c$ is an $n$-bit binary string, denoting the address of the whole cycle, and $i$, $0 \leq i \leq n - 1$, is the offset within the cycle. Two nodes $(i_1, c_1)$ and $(i_2, c_2)$ are adjacent if and only if $c_1 = c_2$ and $i_1 - i_2 \equiv \pm 1 \pmod{n}$ or $i_1 = i_2$ and $c_1$ and $c_2$ differ only in bit $i_1$. Being a 3-regular node-symmetric graph, $CCC_n$ are the sparsest hypercubic network. They are a spanning subgraph of the wrapped butterfly, which is a 4-regular graph. It is known that all sparse hypercubic networks are quasisometric. Cube-connected cycles are weakly-orthogonal: a displacement in a cycle or along a hypercube edge does not change the other coordinates, but from a given node only one hypercube dimension can be traversed. Routing consists in alternating hypercube edges and cycle subpaths. A path from node $a$ to node $b$ will be denoted by $P(a, b)$.

3 Previous work

3.1 All-output-port WH networks

The best known algorithm for $n$-dimensional binary hypercubes has been proposed in [3]. It uses $e$-cube routing and is asymptotically optimal. The paper [1] describes a nearly optimal solution for the hypercube. For $n \leq 31$, the algorithm needs at most 1 extra round with respect to the trivial lower bound $\lceil \frac{n}{\log_2(n+1)} \rceil$, but it does not use $e$-cube routing.

Much effort has been spent on finding optimal algorithms for low-dimensional meshes and tori. The best known algorithm for 2-D tori is described in [8]. It requires at most 2 (5) rounds more than the trivial lower bound $\lceil \log_5 N \rceil$ for $N$-node square (rectangular, respectively) 2-D tori. It uses again dimension-ordered routing. Nearly optimal OAB algorithms for 2-D and 3-D meshes of trees have been given in [6] and [5], respectively.

3.2 Cube-connected cycles

In the all-output-port model with distance-sensitive switching, such as store-and-forward, the trivial flooding algorithm provides an optimal OAB, the number of rounds is bounded by the diameter. The same holds for the 2-output port model.
In the 1-port model, the best known algorithm requires 2 more rounds [2]. Algorithms for all-to-all broadcast in the 1-port combining model have been studied for both half- and full-duplex models in [4].

In 1-port WH $CCC_n$, there is a simple and optimal algorithm. In the first $n$ rounds, each cycle is informed by the standard hypercube algorithm using the spanning binomial tree and the rest of the nodes is informed in $\lceil \log_2 n \rceil$ rounds within cycles.

Paper [7] gives an efficient unicast-based multicast algorithm in 1-port WH $CCC_n$. A multicast within a group of $m$ nodes can be achieved in $\lceil \log_2 m \rceil$ rounds supposing the hardware provides 2 virtual channels per one physical cycle channel. The algorithm is not depth-contention-free.

4 Preliminary results

4.1 The trivial lower bound

Let $\rho_{OAB}(G)$ denote the trivial lower bound on the number of rounds of OAB in the all-output-port WH network $G$.

**Lemma 1**

$$\rho_{OAB}(CCC_n) = \lceil \log_4 |V(CCC_n)| \rceil = \lceil (n + \log_2 n)/2 \rceil.$$  

4.2 OAB in linear arrays and cycles from internal sources

Consider an $N$-node linear array $A$ with nodes labeled consecutively $0, \ldots, N - 1$. Given integers $b \geq 0$ and $c \geq 1$, we define the $(b, c)$-subarray of $A$ to be the part of $A$, starting in node $b$ and having $c$ nodes. The leader of the $(b, c)$-subarray is defined to be the node $leader(b, c) = b + \lceil c/2 \rceil$.

If the source of an OAB is in the middle of a linear array, then the linear array can be split into 3 equal parts (up to 1 node). The source can inform the two boundary parts in 1 round and the same algorithm is applied recursively in all 3 parts in parallel. The algorithm is therefore depth-contention-free. A pseudocode for such a 3-ary decomposition algorithm, called PathOAB, is given in Figure 2. OAB from node $\lceil N/2 \rceil$ is performed by calling PathOAB($0, N$). The algorithm proves constructively the following statement.

**Lemma 2** OAB in an all-output-port WH linear array of $N$ nodes from source node $\lceil N/2 \rceil$ can be done in $\lceil \log_3 N \rceil$ rounds, which is optimal, using a depth-contention-free algorithm.

Since an $N$-node cycle is node-symmetric, we get easily

**Corollary 1** OAB in an all-output-port WH cycle of $N$ nodes from any internal source $s$ can be done in $\lceil \log_3 N \rceil$ rounds, using a depth-contention-free algorithm.

Let us denote the corresponding algorithm by CycleOAB($s, N$).
algorithm PathOAB(start, count)
begin
  $S_{OUT} = \{(start, count)\}$;
  for $i = 1$ to $\lceil \log_3 count \rceil$ do
    begin
      $S_{IN} = S_{OUT}$; $S_{OUT} = \{\}$;
      for all $s = (beg, cnt) \in S_{IN}$ do in parallel
        begin
          sidecnt = cnt div 3;
          if (cnt mod 3 == 2) then sidecnt = sidecnt + 1;
          $S_{OUT} = S_{OUT} + \{(beg, sidecnt), (beg + cnt - sidecnt, sidecnt), (beg + sidecnt, cnt - 2 \cdot sidecnt)\}$;
        end
        if (cnt > 2) then
          begin
            leader(beg, cnt) sends the packet to nodes leader(beg, sidecnt) and leader(beg + cnt - sidecnt, sidecnt);
          end
        else
          if (cnt == 2) then leader(beg, cnt) sends the packet to leader(beg, 1);
      end
    end
end.

Figure 2: Pseudocode of algorithm PathOAB.

4.3 OAB in linear arrays and cycles from external sources

Our algorithm for OAB in $CCC_n$ will induce OABs in cycles from external sources. Let us start with linear arrays.

**Lemma 3** OAB in an all-output-port WH linear array of $N$ nodes, where the source is an external node connected to node 0, can be done within $\lceil \log_3(2N + 1) \rceil$ rounds, which is optimal, using a depth-contention-free algorithm.

**Proof.** The lower bound on the number of rounds is as follows: Let $t_i$ be the number of informed nodes after round $i$. Then $t_0 = 0$ and for $i \geq 1$, $t_i = 3 \cdot t_{i-1} + 1$, since each internal informed node can inform 2 new nodes and the external node can inform 1 new node. The least $i$ such that $t_i = \frac{3^i - 1}{3 - 1} = N$ is $i = \lceil \log_3(2N + 1) \rceil$. Note that $\lceil \log_3(2N + 1) \rceil \leq 1 + \lceil \log_3 N \rceil$. The pseudocode of the algorithm ExtPathOAB, which proves constructively the lemma, is described in Figure 3. It has one parameter count, which is again to be substituted by the linear array size, i.e., $N$. The external node disseminates the information
in the ever diminishing subarrays taken from the main linear array from the right to the left towards the node 0, whereas in the parts with informed leaders, the algorithm PathOAB from Figure 2 completes the broadcast. Figure 4 shows 3 rounds of ExtPathOAB(13). Similarly to the algorithm PathOAB, the algorithm ExtPathOAB is depth-contention-free.

A similar approach applies to cycles, which are node-symmetric.

**Corollary 2** OAB in an all-output-port WH cycle of N nodes, where the source is an external node connected to a cycle node x, can be done within \([\log_3(2N + 1)]\) rounds.

Let us denote the corresponding OAB algorithm by ExtCycleOAB\(x, N\). By applying Lemma 3 twice, we get

**Corollary 3** OAB in an all-output-port WH cycle of N nodes, where N is even (odd), from 2 external sources connected to nodes x and \((x + [N/2]) \mod N\) of the cycle, can be done in \([\log_3(N + 1)]\) \((\lceil \log_3(N + 2) \rceil\), respectively) rounds.

Let us denote the corresponding algorithm by Ext2CycleOAB\(x, N\).
Figure 4: The whole broadcast tree of the 3-round ExtPathOAB(13).

5 The OAB algorithm in \( \text{CCC}_n \)

Our main result is a nearly optimal OAB algorithm in an all-output-port WH \( \text{CCC}_n \). The algorithm depends on the parity of \( n \) and it is described by pseudocodes in Figures 5 and 7, respectively. In both cases, we use standard minimal routing in \( \text{CCC}_n \). The corresponding routing function is denoted by \( R \). If there are 2 shortest paths for a given source-destination pair, the algorithm always chooses one of them by deciding the output port, so that all paths starting from a given source in a given round are minimal and link-disjoint.

Since \( \text{CCC}_n \) are node-symmetric, we assume w.l.o.g. that the source is node \( (0,0)^n \).

Definition 1 Let \( n \geq 3, 0 < i < n \), and let \( s = (0,c) \) be the node of \( \text{CCC}_n \), \( c \in \mathcal{B}^n \). Then \( \text{dst}(s,i,n) \) is defined as a set of 3 nodes:

\[
\text{dst}(s,i,n) = \{(0,\text{neg}_i(\text{neg}_0(c))), (0,\text{neg}_{n-i}(\text{neg}_0(c))), (0,\text{neg}_{n-i}(c))\}.
\]

5.1 \( n \) is odd

Theorem 1 If \( n \) is odd, then the algorithm OddCccOAB(n) whose pseudocode is in Figure 5 completes a OAB in an all-output-port WH \( \text{CCC}_n \) in \( (n - 1)/2 + \lfloor \log_3(2n + 1) \rfloor \) rounds. Moreover, it is a depth-contention-free algorithm.

Proof. Let us prove first that at the end of Phase 1, in exactly one half of cycles of \( \text{CCC}_n \), there is exactly one informed leader and exactly one half of cycles is uninformed. More specifically, let us prove that the cardinality of \( S \), the set of all informed leaders, at the end of Phase 1 is \( 2^{n-1} \) and

\[
\forall w \in \mathcal{B}^{n-1} \exists \alpha \in \mathcal{B} \text{ such that } (0, w\alpha) \in S \text{ and } (0, w\overline{\alpha}) \notin S. \tag{1}
\]

(1) can be proved inductively by proving that at the end of round \( j \), \( 1 \leq j \leq \frac{n-1}{2} \), of Phase 1

\[
\forall x, y \in \mathcal{B}^j \exists \alpha \in \mathcal{B} \text{ such that } (0, 0^j x y \alpha \overline{\alpha}) \in S \text{ and } (0, 0^j x y \overline{\alpha}) \notin S. \tag{2}
\]
Figure 5: Pseudocode of algorithm OddCccOAB.

where \( k = \frac{(n - 1)}{2} - j \). This statement holds at the end of round
\( j = 1 \), since \( S = \{(0,0^0)\} \cup \text{dss}(0,0^0,\frac{n-1}{2},n) = \{(0,0^{\frac{n-1}{2}-1}1000^{\frac{n-1}{2}-1}0),
(0,0^{\frac{n-1}{2}-1}1010^{\frac{n-1}{2}-1}1), (0,0^{\frac{n-1}{2}-1}110^{\frac{n-1}{2}-1}0), (0,0^{\frac{n-1}{2}-1}100^{\frac{n-1}{2}-1}0)\} \).

Let us assume that at the end of round \( j < \frac{n-1}{2} \), (2) holds and let us consider
any \( s = (0,z) \in S, z = 0^{\frac{n-1}{2}-1}xy0^{\frac{n-1}{2}-2}x^{\alpha} \). After round \( j + 1 \), \( \text{dss}(s,\frac{n-1}{2}-j,n) \) is
added to \( S \) due to \( s \).
\[
\text{dss}(s,\frac{n-1}{2}-j,n) = \\
\{(0,\text{neg}_{\frac{n-1}{2}-j}(\text{neg}_0(z))), (0,\text{neg}_{\frac{n-1}{2}-j}(\text{neg}_{\frac{n-1}{2}-j-1}(z))), (0,\text{neg}_{\frac{n-1}{2}-j}(\text{neg}_{\frac{n-1}{2}-j}(z))), \\
(0,0^{\frac{n-1}{2}-1}xy10^{\frac{n-1}{2}-1}x^{\alpha}, (0,0^{\frac{n-1}{2}-1}xy10^{\frac{n-1}{2}-1}1x^{\alpha})
\}
\]

Hence, \( \{s\} \cup \text{dss}(s,\frac{n-1}{2}-j,n) = \{(0,0^{\frac{n-1}{2}-(j+1)}\beta xy0^{\frac{n-1}{2}-(j+1)}\alpha') \mid \beta, \gamma \in \mathcal{B}\} \), where \( x, y \in \mathcal{B}' \) are given and \( \alpha' = \alpha \) if \( \beta = 0 \) and \( \gamma = 1 \) and \( \alpha' = \alpha \) otherwise.
Since this argument applies to any \( s \in S \), the induction step is proved. This proves
(1).

In Phase 2, each informed leader \( (0,w\alpha) \in S \) becomes the source of a OAB
in its own cycle \( (*,w\alpha) \) and simultaneously, it becomes the external source of a
OAB in its "complementary" uninformed cycle (\(*, w, \alpha\)).

Due to Corollaries 1 and 2, Phase 2 is depth-contention-free. Let us show that Phase 1 is depth-contention-free, too. This follows from two facts:

1. All minimal paths going from all nodes \(s = (0, z) \in S\) to corresponding nodes in \(d sr(s, \frac{z-1}{2} - (j + 1), n)\) in round \(j\) are node-disjoint. The path \(P((0, z), (0, neg \frac{z-1}{2} - (neg_0(z))))\) starts by 1 hypercube edge, continues by several cycle edges and 1 hypercube edge, and ends by several cycle edges. The path \(P((0, z), (0, neg \frac{z-1}{2} - (neg_0(z))))\) starts by cycle edges, continues by 1 hypercube edge, several cycle edges, and 1 more hypercube edge, and ends by several cycle edges. The path \(P((0, z), (0, neg \frac{z-1}{2} - (neg_0(z))))\) starts by cycle edges, traverses a hypercube edge, and ends by cycle edges.

2. Any path starting in node \((0, 0^k x y 0^k \alpha) \in S, x, y \in B^l, k = \frac{z-1}{2} - j\) in subsequent rounds traverses only nodes \((l, c)\), where \(c = \chi xy \psi \alpha\) or \(c = \chi xy \psi \alpha\), where \(\chi, \psi \in B^k\), and therefore, any 2 paths originated in 2 different leaders.
in any 2 successive rounds are node-disjoint.

Figure 6 shows 3 rounds of OddCccOAB(3). Phase 1 has only 1 round and Phase 2 has 2 rounds.

5.2 \( n \) is even

```
algorithm EvenCccOAB(\( n \))
/* \( n \) is the dimension of CCC */
/* the source is (0, 0) */
begin
    \( S = \{(0, 0^n)\} \);
    /* Phase 1: rounds 1 */
    for \( i = \frac{n}{2} - 1 \) downto 1 do
        begin
            for all \( s \in S \) do
                in parallel
                \( s \) sends the packet to the set of nodes dst(\( s, i, n \));
                \( S = S + \bigcup_{s \in S} \) dst(\( s, i, n \));
        end
    /* Phase 2a: round \( n/2 \) */
    for all \( s = (0, x0y\alpha) \in S, x, y \in g^{n/2-1} \), do_in_parallel
        \( (0, x0y\alpha) \) sends the packet to the nodes \( (n/2, x1y\alpha) \) and \( (n/2, x1y\overline{\alpha}) \);
    /* Phase 2b: the rest of rounds */
    for all \( s = (0, x0y\alpha) \in S, x, y \in g^{n/2-1} \), do_in_parallel
        parbegin
            apply algorithm CycleOAB(0, \( n \)) in cycle \( (s, x0y\alpha) \);
            /* i.e., OAB with source \( (0, x0y\alpha) \) */
            apply algorithm CycleOAB(\( n/2 \)) in cycle \( (s, x1y\alpha) \);
            /* i.e., OAB with source \( (n/2, x1y\alpha) \) */
            apply algorithm CycleOAB(\( n/2 \)) in cycle \( (s, x1y\overline{\alpha}) \);
            /* i.e., OAB with source \( (n/2, x1y\overline{\alpha}) \) */
            apply algorithm Ex2CycleOAB(0, \( n \)) in cycle \( (s, x0y\overline{\alpha}) \) with external sources \( (0, x0y\alpha) \) and \( (n/2, x1y\overline{\alpha}) \);
        parend
    end.
```

Figure 7: Pseudocode of algorithm EvenCccOAB.

**Theorem 2** If \( n \) is even, then the algorithm EvenCccOAB(\( n \)) whose pseudocode is in Figure 7 completes a OAB in an all-output-port WH CCC in \( n + |\log_3(n + 1)| \) rounds. Moreover, it is a depth-contention-free algorithm.
Proof. Similar to the proof of Theorem 1. At the end of Phase 1, in exactly one quarter of cycles of \( CCC_n \), there is exactly one informed leader. This follows from the fact that after Phase 1

\[
\forall x, y \in B^{n/2} \exists \alpha \in B \text{ such that } (0, x0y\alpha) \in S.
\]

This can again be proved inductively by showing that after round \( j \) of Phase 1,

\[
\forall x, y \in B^{j/2} \exists \alpha \in B \text{ such that } (0, 0^jx0y0^i\alpha) \in S \text{ and } (0, 0^jx0y0^i\overline{\alpha}) \notin S.
\]

where \( k = n/2 - 1 - j \). In the first round of Phase 2, each leader informs 2 further nodes so that \( \forall x, y \in B^{n/2} \forall \alpha \in B \), cycle \( (\ast, x1y\alpha) \) has one informed leader and \( \forall x, y \in B^{n/2} \exists \alpha \in B \) such that cycle \( (\ast, x0y\alpha) \) has one informed leader, whereas cycle \( (\ast, x0y\overline{\alpha}) \) has none. In the rest of rounds of Phase 2, all cycles with informed leaders perform OABs using algorithm \( CycleOAB(n) \) and at the same
time, uninformed cycles perform OABs initiated by two external sources, adjacent to nodes 0 and \( n/2 \) in these cycles, respectively.

Figure 8 shows a part of the broadcast tree of EvenCccOAB(6) related to the source \((0,0^6)\). Phase 1 consists of 2 rounds and Phase 2 consists of \( 1 + 2 \) rounds.

Since the Odd(Even)CccOAB algorithm uses the minimal routing \( R \), we will shortly call it the \( R \)-algorithm.

6 Deadlock avoidance

The \( R \)-algorithm is depth-contention-free and so, a deadlock cannot appear even if the broadcast tree grows asynchronously. However, the minimal routing \( R \) is not deadlock-free since the minimal routing even in cycles is not deadlock-free. If 2 asynchronous OABs with different sources run concurrently or if a OAB proceeds concurrently with some other communication conforming with \( R \), a deadlock can appear, since the channel dependency graph for CCC\(_n\) and \( CDG(CCC_n, R) \), is not acyclic.

In this section, we propose a solution. Each full-duplex physical channel is formed by two antiparallel directed links. By restricting \( R \), we define another routing function \( R' \), based on the \( e \)-cube routing. However, to keep the network CCC\(_n\) connected, we only need in one direction, called down, 2 virtual links A and B per each cycle link. These requirements are smaller than requirements for the HC routing introduced in [7]. They need 2 virtual links in both directions.

Definition 2 Let \( 0 \leq i \leq n - 1 \) and \( \beta \in B^n \). We label links of CCC\(_n\) as follows:

- cube\(_i \) - the link from \((i, \beta)\) to \((i, \neg \beta(\beta))\),
- up\(_i \) - the link from \((i, \beta)\) to \(((i + 1) \mod n, \beta)\),
- down\(_i^A \) - the virtual link A from \(((i + 1) \mod n, \beta)\) to \((i, \beta)\),
- down\(_i^B \) - the virtual link B from \(((i + 1) \mod n, \beta)\) to \((i, \beta)\).

We define the ordering of the links as follows:

\[
\text{down}_{n-1}^A < \text{down}_{n-2}^A < \ldots < \text{down}_0^A < \\
< \text{cube}_0 < \text{up}_0 < \text{cube}_1 < \text{up}_1 < \ldots < \text{cube}_{n-1} < \text{up}_{n-1} < \\
< \text{down}_{n-1}^B < \text{down}_{n-2}^B < \ldots < \text{down}_0^B.
\]

Figure 9 shows this ordering within a cycle of CCC\(_n\).

Let \( R' \) be the routing function that generates only paths in strictly increasing order of the link labels. Since the \( CDG(CCC_n, R') \) does not contain cycles, \( R' \) is deadlock-free. Each path from \((a, \alpha)\) to \((b, \beta)\) conforming with \( R' \) consists of up to three subpaths:
Figure 9: Ordering of links within a cycle of $CCC_n$. Each edge of a cycle is replaced by one $up$ link and two $down$ links.

1. The subpath from source node $(a, \alpha)$ to node $(i, \alpha)$, where $i$ is the lowest bit in which $\alpha$ and $\beta$ differ. This subpath can use all $down^A$ links and $up_x$, $0 \leq x < i$, links. Hence, these subpaths are not guaranteed to be minimal.

2. The subpath from node $(i, \alpha)$ to node $(j, \beta)$, where $j$ is the highest bit in which $\alpha$ and $\beta$ differ. This subpath traverses dimensions in increasing order and it can use $cube_x$, $i \leq x \leq j$, links and $up_x$, $i \leq x < j$, links.

3. The subpath from node $(j, \beta)$ to node $(b, \beta)$. This subpath can use all $down^B$ links and $up_x$, $j \leq x < n$, links. Again, this subpath is not guaranteed to be minimal.

**Theorem 3** There exists a OAB $R'$-algorithm in an all-output-port WH $CCC_n$, whose number of rounds is $1 + \frac{2n^2}{2} + \lfloor \log_3(n+2) \rfloor$ rounds for odd $n$ and $1 + \frac{2n^2}{2} + \lfloor \log_3(2n+1) \rfloor$ for even $n$. Moreover, it is a depth-contention-free algorithm.

**Proof.** For any source $(0, \beta)$, $\beta \in \mathbb{B}^n$, the $R'$-algorithm, the OAB algorithm conforming with $R'$, has exactly the same pseudocode as the $R$-algorithm in Section
5. The routing function $R'$ generates the same paths as routing function $R$ in case of such a source.

Consider a OAB $R'$-algorithm with source $(i, \beta)$, $i \neq 0$. It can be obtained by a slight modification of the $R$-algorithm. In the first round, source $(i, \beta)$ sends the packet to nodes $(0, \beta)$ and $(0, \neg x_{[n/2]}(\beta))$ using $R'$. After that, we can view $CCC_n$ as split into 2 halves, which are almost isomorphic to $CCC_{n-1}$, except that each cycle has one abundant node. Now, in both $CCC_{n-1}$-like halves in parallel, we can perform OABs using a minor modification of the $R$-algorithm from Section 5, respecting the fact that cycles have $n$ nodes instead of $n - 1$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\log_2 n$</th>
<th>$\left\lceil \frac{\log_2 n}{2} \right\rceil$</th>
<th>the # of rounds of $R$-algorithm</th>
<th>the # of rounds of $R'$-algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.585</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2.000</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2.322</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2.585</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2.807</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>3.000</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>3.170</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>3.322</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3.459</td>
<td>8</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>3.585</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>3.700</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>3.807</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>3.906</td>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>78</td>
<td>6.285</td>
<td>43</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>79</td>
<td>6.303</td>
<td>43</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>80</td>
<td>6.322</td>
<td>44</td>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>81</td>
<td>6.340</td>
<td>44</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>121</td>
<td>6.919</td>
<td>64</td>
<td>65</td>
<td>66</td>
</tr>
<tr>
<td>122</td>
<td>6.931</td>
<td>65</td>
<td>66</td>
<td>67</td>
</tr>
<tr>
<td>123</td>
<td>6.942</td>
<td>65</td>
<td>67</td>
<td>67</td>
</tr>
<tr>
<td>124</td>
<td>6.954</td>
<td>66</td>
<td>67</td>
<td>68</td>
</tr>
</tbody>
</table>

Table 1: The time complexity of our OAB algorithms in $CCC_n$.

7 Conclusions

The time complexity of the presented algorithms is evaluated in Table 1. The greatest $n$ when the $R$-algorithm is optimal is 80. The smallest $n$ such that the
difference between the trivial lower bound and the number of rounds of the $R$-algorithm is greater than 1 is 123. Note that the $CCC_{123}$ have around $1.3 \cdot 10^{39}$ nodes. The deadlock-free $R'$-algorithm needs at most 1 round more than the $R$-algorithm.

Since $CCC_n$ are a spanning subgraph of the $n$-dimensional wrapped butterfly $wBF_n$, our algorithms performing a OAB in $CCC_n$ can run without any modification on $wBF_n$ in $\rho_{OAB}(wBF_n) \cdot \log_4 5$ rounds.

References


Petr Salinger is a Ph.D. candidate at the Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Karlovo nám. 13, 121 35, Praha 2, Czech Republic. E-mail: salinger@sun.felk.cvut.cz
Pavel Tvrdik is with the Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Karlovo nám. 13, 121 35, Praha 2, Czech Republic. E-mail: tvrdik@sci.felk.cvut.cz
New Protocols for Asymmetric Communication Channels

JOHN WATKINSON
Modelwire Inc., USA

MICAH ADLER
University of Massachusetts, USA

FAITH E. FICH
University of Toronto, Canada

Abstract
In this paper, we study the problem of sending an n-bit binary string drawn from a probability distribution \( D \) across a communication channel connecting a client and a server, where the bandwidth from the client to the server is much smaller than the bandwidth from the server to the client. We assume that the client knows the string \( x \) but not the distribution \( D \), and the server knows the distribution \( D \), but not the string \( x \). Adler and Maggs demonstrated protocols for this problem where the expected number of bits sent by the client is \( O(\frac{H(D)}{B4}) \) and the expected number of bits sent by the server is \( O(\frac{n}{B4}) \). Here, \( H(D) \) is the binary entropy of the distribution and a lower bound on the expected number of bits that the client must send. In this paper, a new protocol is presented in which the expected number of bits sent by the client is \( H(D) + 2 \), and the expected number of bits sent by the server is \( n(H(D) + 2) \). This protocol is then generalized so as to reduce the number of rounds of communication at the expense of computation and bits sent by the server.

Keywords
asymmetric communication protocols, interactive source coding, expected bit complexity

1 Introduction
Recently, many networking technologies have been introduced that are asymmetric in terms of the bandwidth offered in the two directions of transmission. For example, many high-speed home Internet connectivity packages such as cable...
modems and asymmetric digital subscriber lines (ADSL) have much faster download speeds than upload speeds. Most satellite Internet connections offer download speeds on the order of megabits per second, but rely on a conventional telephone modem for uploads with speeds on the order of 10 kilobits per second.

These new networking technologies lead to the following question: what is the best way to use the high bandwidth direction of an asymmetric communication channel to improve the performance of the low bandwidth direction? This question was first addressed in [1]. That work assumes that two parties, called the client and the server, are connected by a channel with much higher bandwidth from the server to the client than from the client to the server. They present a number of protocols that use the high-speed link to reduce the number of bits that the client must send on the slower link. Specifically, in many situations, the client is able to inform the server of an \( n \)-bit string \( x \) by sending significantly fewer than \( n \) bits.

The protocols from [1] are within a constant factor of optimal in terms of the number of bits sent by the client, as well as the total number of bits sent by both parties. In this paper, we address the following question: How small can we make the number of bits sent by the client, without significantly increasing the total number of bits sent? We introduce two protocols that address this question. The first is Bit-Efficient-Split, a protocol where the number of bits sent by the client is within a small additive constant of the optimal. However, the number of bits sent by the server is increased and the client only sends one bit per round of communication. The second protocol, Round-Efficient-Split, is a generalization of Bit-Efficient-Split. It reduces the number of rounds of communication at the expense of server computation and the number of bits sent by the server.

1.1 Communication Model

The asymmetric communication model proposed by Adler and Maggs in [1] is used in this paper. The model consists of a client and a server connected by a communication link. The bandwidth from the server to the client is greater than the bandwidth from the client to the server. The client holds a string \( x \) drawn from a probability distribution \( D \) on the set of all \( n \)-bit strings. The string \( x \) is unknown to the server and the distribution \( D \) is unknown to the client. The client must inform the server of the string \( x \).

The computational effort on the part of the server is modelled by restricting how the server can access the distribution and counting the number of accesses the server performs. Specifically, an access to the distribution is a black box query. It consists of a binary string \( t \) of length \( m \leq n \). The result of the query is the total probability of all \( n \)-bit strings that have \( t \) as a prefix.

A round of communication consists of the server sending some non-zero number of bits to the client, followed by the client sending some non-zero number of bits back to the server.
In this model, the performance of a protocol is characterized by the tuple $[\sigma, \phi, \lambda, \rho]$, where $\sigma$ is the expected number of bits sent by the server, $\phi$ is the expected number of bits sent by the client, $\lambda$ is the expected number of black box queries performed by the server, and $\rho$ is the expected number of rounds of communication. All expectations are with respect to the random variable $x$ chosen from $D$. In all the protocols presented, the client’s computational effort is minimal, and so is ignored.

1.2 Previous Work and Results

Given a distribution $D$ over $n$-bit strings, the entropy of $D$ is

$$H(D) = \sum_{x \in D} D(x) \log_2 \frac{1}{D(x)},$$

where $x \in D$ means that $D(x) > 0$. Shannon’s well-known Coding Theorem [5] states that even if the client is aware of the distribution $D$, the expected number of bits sent by the client must be at least $H(D)$. Thus, when the client is unaware of the distribution, the expected number of bits sent by the client must still be at least $H(D)$.

In [1], Adler and Maggs study the communication model in detail and demonstrate some lower bounds on the number of client bits, the number of server bits, server computation and the number of rounds. They also present three protocols. The first is Computation-Efficient. Their analysis demonstrates that this protocol has performance characteristics $[3n, 1.71H(D) + 1, 3n, 1.71H(D) + 1]$, and this analysis has been improved to $[3n, 1.089H(D) + 1, 3n, 1.089H(D) + 1]$ by [3]. The second is Round-Efficient, a constant-round protocol that requires considerable computation on the part of the server. Their original analysis gives performance characteristics $[O(n), O(H(D) + 1), 2^n, 6]$, which has been improved to $[O(n), 2H(D) + O(1), 2^n, 3]$ in [6]. Adler and Maggs also present a hybrid of these two protocols, Computation-Rounds-Tradeoff($c$), with performance characteristics $[O(n), O(H(D) + 1), O(2^n), O(\min(\frac{c}{\epsilon}, H(D) + 1)),$ for any positive integer $c \leq n$. For all these protocols, the expected number of bits sent by the client is within a multiplicative factor of optimal. We also mention that [4] provides a different technique such that the expected number of client bits is only $1.47H(D)$. However, this result has been subsumed by the improved analysis of [3] on the protocols of [1].

The two communication protocols presented in this paper improve the expected number of bits sent by the client to be within an additive constant of optimal. In Section 2.2, we present Bit-Efficient-Split, a protocol with performance characteristics $[nH(D) + 2, H(D) + 2, nH(D) + 3, H(D) + 2]$. In Section 2.3, we present Round-Efficient-Split, a generalization of Bit-Efficient-Split, with performance characteristics $[O(2^k nH(D)), H(D) + 2, O(2^k nH(D)), \frac{H(D) + 1}{2} + 2]$.  

In Section 2.2, we present Bit-Efficient-Split, a protocol with performance characteristics $[nH(D) + 2, H(D) + 2, nH(D) + 3, H(D) + 2]$. In Section 2.3, we present Round-Efficient-Split, a generalization of Bit-Efficient-Split, with performance characteristics $[O(2^k nH(D)), H(D) + 2, O(2^k nH(D)), \frac{H(D) + 1}{2} + 2]$. 

In [1], Adler and Maggs study the communication model in detail and demonstrate some lower bounds on the number of client bits, the number of server bits, server computation and the number of rounds. They also present three protocols. The first is Computation-Efficient. Their analysis demonstrates that this protocol has performance characteristics $[3n, 1.71H(D) + 1, 3n, 1.71H(D) + 1]$, and this analysis has been improved to $[3n, 1.089H(D) + 1, 3n, 1.089H(D) + 1]$ by [3]. The second is Round-Efficient, a constant-round protocol that requires considerable computation on the part of the server. Their original analysis gives performance characteristics $[O(n), O(H(D) + 1), 2^n, 6]$, which has been improved to $[O(n), 2H(D) + O(1), 2^n, 3]$ in [6]. Adler and Maggs also present a hybrid of these two protocols, Computation-Rounds-Tradeoff($c$), with performance characteristics $[O(n), O(H(D) + 1), O(2^n), O(\min(\frac{c}{\epsilon}, H(D) + 1)),$ for any positive integer $c \leq n$. For all these protocols, the expected number of bits sent by the client is within a multiplicative factor of optimal. We also mention that [4] provides a different technique such that the expected number of client bits is only $1.47H(D)$. However, this result has been subsumed by the improved analysis of [3] on the protocols of [1].

The two communication protocols presented in this paper improve the expected number of bits sent by the client to be within an additive constant of optimal. In Section 2.2, we present Bit-Efficient-Split, a protocol with performance characteristics $[nH(D) + 2, H(D) + 2, nH(D) + 3, H(D) + 2]$. In Section 2.3, we present Round-Efficient-Split, a generalization of Bit-Efficient-Split, with performance characteristics $[O(2^k nH(D)), H(D) + 2, O(2^k nH(D)), \frac{H(D) + 1}{2} + 2]$.
2 The Protocols

Consider the lexicographic ordering of binary strings, where \( x < y \) if \( x \) is a prefix of \( y \) or if \( z0 \) is a prefix of \( x \) and \( z1 \) is a prefix of \( y \), for some string \( z \). Then \( x \leq y \) if and only if \( x < y \) or \( x = y \).

The protocols work by having the server send carefully chosen strings to the client, who indicates whether they are lexicographically greater than the client’s string \( x \). We begin, in Section 2.1, by defining these strings and describing how to find them. The algorithms **Bit-Efficient-Split** and **Round-Efficient-Split** are presented in the following two sections.

2.1 Split Strings

Let \( F(x) = \sum \{D(y) \mid y \text{ is an } n\text{-bit string and } y < x \} \) denote the cumulative distribution function for a distribution \( D \) over \( n\)-bit strings. Note that \( D(x) \) is not included in this sum. Plotting \( F(x) \) versus \( x \) yields a rising staircase of steps of varying
height. The step corresponding to \( x \) has height \( D(x) \). Let \( F'(x) = F(x) + \frac{1}{2}D(x) \). This value is called a split point for \( D \). The functions \( F \) and \( F' \), for a particular distribution over 3-bit strings, are illustrated in Figure 1. Given a real number \( v \in (0, 1] \), let \( t \) be the lexicographically greatest \( n \)-bit string such that \( F'(t) \leq v \). We call \( t \) the \( v \)-split string for \( D \). Note that, if \( F'(0^n) > v \), then there is no \( v \)-split string.

One way to find the \( v \)-split string for \( D \) is to first find the string \( z^* \), defined to be the lexicographically greatest string such that \( F(z^*) \leq v \). This can be done by using the results of black box queries to effectively perform binary search. Then the \( v \)-split string is either \( z^* \) or its predecessor. Specifically, if \( F'(z^*) \leq v \), then \( z^* \) is the \( v \)-split string. Otherwise, the predecessor \( y \) of \( z^* \), if it exists, is the \( v \)-split string, since \( F'(y) = F(y) + \frac{1}{2}D(y) \leq F(y) + D(y) = F(z^*) \leq v \). See Figure 2 for examples of split strings. An algorithm, Split-String, finds the \( v \)-split string.

Algorithm Split-String

1. Initialize \( z \) to the empty string and real numbers \( p \) and \( r \) to 0 and 1, respectively.
2. While the length of \( z \) is less than \( n \):
   - Let \( q = D(z0) \).
   - If \( p + q \geq v \) then append 0 to \( z \) and \( r \leftarrow p + q \).
   - Otherwise, append 1 to \( z \) and \( p \leftarrow p + q \).
3. Let \( s \leftarrow (r + p)/2 \). This is equal to \( F'(z^*) \).
4. If \( s \leq v \) then output \( z \).
5. Otherwise \( s > v \). Two cases remain:
   - If \( z = 0^n \), then indicate that there is no \( v \)-split string and terminate.
   - Otherwise, output the predecessor of \( z \).

There are three loop invariants: \( p = \sum \{D(y) | y \text{ is an } n \text{-bit string and } y < z \} \), \( r = \sum \{D(y) | y \text{ is an } n \text{-bit string and } y \leq z1^n[0,1] \} \), and \( z \) is a prefix of \( z^* \). Thus, when the loop terminates, \( z = z^* \), \( p = F(z^*) \), \( r - p = D(z^*) \), and \( s = (r + p)/2 = F(z^*) + \frac{1}{2}D(z^*) = F'(z^*) \). Each iteration of the loop, one black box query is performed, for a total of \( n \) black box queries. The proof of correctness follows.
2.2 Bit-Efficient-Split

In this protocol, the server performs binary search for the client’s string $x$ by sending split strings to the client. The client responds with 0 if its string is less than or equal to the split string, and 1 if its string is greater than the split string. The server maintains two $n$-bit strings $y$ and $z$ such that $y < x < z$ and two nonnegative numbers $u < v$ such that $F'(x') \in (u, v]$ if and only if $y < x' \leq z$. The complete algorithm **Bit-Efficient-Split** appears below.

During each iteration of the loop, the server determines the $w$-split string, for $w = (u + v)/2$. Suppose that, in some iteration of the loop, a $w$-split string did not exist. Then $F'(0^n) > w$. Since $F'(0^n) \geq D(0^n) = 2F'(0^n) > 2w > v$ and $F'(z) \leq v$, it follows that $z = 0^n$. Hence $y = z$ and the iteration would not have occurred. Therefore, in each iteration, a $w$-split string $t$ exists. Moreover, since $F'(t) \leq w \leq v$, and $F'(x') \leq v$ implies that $x' \leq z$, we know that $t \leq z$.

Since $t$ is the $w$-split string, $F'(x') \leq w$ if and only if $x' \leq t$. If $x \leq t$, then the client sends 0 to the server, who updates $u$ to $w$ and $z$ to $t$. Otherwise, $t < x$ and the client sends 1 to the server. In this case, the server updates $u$ to $w$ and $y$ to be the successor of $t$, so $y < x$. Furthermore, $F'(x') > w$ if and only if $x' > y$. Thus, in both cases, the invariants remain true.
Algorithm Bit-Efficient-Split

1. The server initializes $u$ to 0, $v$ to 1, and $z$ to $1^n$.
2. If $D(0^n) = 0$, the server computes the 0-split string $t_0$ and initializes $y$ to the successor of $t_0$;
3. otherwise, the server initializes $y$ to $0^n$.
4. While $y \neq z$ do:
   - Let $w = \frac{u + v}{2}$. The server computes the $w$-split string $t$. It sends $t$ to the client.
   - If $x \leq t$, then the client sends 0 to the server. Otherwise, the client sends 1.
   - If the server receives 0, then $v \leftarrow w$ and $z \leftarrow t$. Otherwise, $u \leftarrow w$ and $y \leftarrow$ the successor of $t$.
5. The server outputs $z$.

Theorem 1 For any distribution $D$, Bit-Efficient-Split has performance characteristics $\left[ n \frac{H}{D} + 2, H(D) + 2, n(H(D) + 3), H(D) + 2 \right]$.

Proof. We begin by deriving an upper bound on the number of rounds performed by the protocol as a function of $D(x)$, where $x$ is the client’s string. Note that $D(x) > 0$. Let $m = \left\lfloor \log_2 \frac{1}{D(x)} \right\rfloor + 1$ and suppose that the protocol performs at least $m$ rounds. Let $u_m$ and $v_m$ denote the values of $u$ and $v$, respectively, at the end of round $m$. The interval $(u_m, v_m]$ has length $2^{-m}$, since the interval $(u, v]$ initially has length 1 and decreases by half each iteration.

By the loop invariants, the interval $(u_m, v_m]$ contains $F^\ell(x)$. Since $D(x) \geq 2^{1-m}$ and $F^\ell(x)$ is the midpoint of the interval $(F(x), F(x) + D(x)]$, this interval must entirely contain $(u_m, v_m]$. Thus, $y = z = x$ at the end of the round and the protocol terminates. Hence, the number of rounds is at most $m + 1 = \left\lfloor \log_2 \frac{1}{D(x)} \right\rfloor + 1 \leq \log_2 \frac{1}{D(x)} + 2$ and the expected number of rounds is at most:

$$
\sum_{x \in D} D(x) \left( \log_2 \frac{1}{D(x)} + 2 \right)
= \left( \sum_{x \in D} D(x) \log_2 \frac{1}{D(x)} \right) + 2
= H(D) + 2.
$$
Since the client sends one bit each round, the above bound also applies to the expected number of bits sent by the client. Each round, the server sends an $n$-bit string to the client, so it follows that the expected number of bits that the server sends is $n(H(D) + 2)$. **Split String** uses at most $n$ black box queries to find a split string. Since at most $H(D) + 3$ split strings are required (one before the main loop of the protocol and $H(D) + 2$ during the main loop), the expected number of black box queries is at most $n(H(D) + 3)$.

Notice that the server could compute a split string that has been sent to the client during an earlier round of the protocol. This corresponds to the situation where the interval $[u, w]$ does not contain any split points. A more efficient implementation of the protocol would skip such rounds.

This protocol performs an “interactive” version of Shannon-Fano-Elias coding of $x$ (as discussed in [2]). The string formed by the concatenation of the bits sent by the client is the Shannon-Fano-Elias codeword for $x$.

### 2.3 Protocol Round-Efficient-Split($k$)

In the protocol **Bit-Efficient-Split**, the server maintains a subinterval of $[0, 1]$ that contains $F^i(x)$. Each round, one half of the interval is removed depending on the bit received by the client. While this is very efficient with respect to the number of bits sent by the client, the client is only sending one bit during each round. Typically, there is latency introduced with each round-trip between the server and the client, so conducting as few rounds as possible is desirable.

A simple modification to **Bit-Efficient-Split** is to have the server subdivide its interval into $2^k$ equal size subintervals and to send to the client, for each subinterval $[u, v]$, the **split string for the subinterval**, i.e. the $v$-split string. The client can then simulate $k$ rounds of the **Bit-Efficient-Split** protocol, with the $k$ single-bit responses sent back to the server in a single batch. The server is required to send $2^k$ split strings each round with this protocol. Although large, this may be a viable trade-off to reduce the number of rounds by approximately a factor of $k$.

A difficulty with this method is that the total number of bits sent by the client must be a multiple of $k$. This may introduce an additive constant of $k$ to the number of bits sent by the client. The additional bits sent by the client occur when the client’s string has large probability relative to the subinterval under consideration by the server during the last round of the protocol. As a result, the server may send several copies of the string to the client, to which the client replies several times. Alternatively, the client could send back the number of split strings (among the $2^k$ that were sent) that are lexicographically greater than or equal to $x$. This also uses $k$ bits each round.

To limit the total number of bits sent by the client to $H(D) + 2$, **Round-Efficient-Split($k$)** requires some additional ideas.

A string $s$ is a **span string** for an interval $I$ if the interval $(F(s), F(s) + D(s)]$ contains $I$. Furthermore, if $F(x) \in I$, then $x = s$. In our algorithm, the server deter-
mines a representative string for each of the $2^k$ subintervals of the current interval. The representative string is either the span string for the interval, if it exists, or the split string for the interval. Note that, if an interval $(u, v]$ has no split string, then $F'(0^v) > v$. This implies that $0^v$ is a span string for the interval $(u, v]$, since $F(0^v) = 0 \leq u$.

During a round of the protocol, the server sends all these strings to the client, and indicates, for each, whether it is a span string or a split string. If the client’s string $x$ is one of the span strings, the client indicates which one. Otherwise, the client indicates the least split string that is greater than $x$ and further rounds ensue.

The representative string for each interval $(u, v]$ can be computed using a slight variant of the Split String algorithm. Since $p = F(z)$ and $r = F(z) + D(z)$, it
suffices to insert the following line between parts 2 and 3:

2.5 If \( p \leq u \) and \( r \geq v \), then indicate that \( z \) is the span string for the interval \((u, v)\) and terminate.

Note that a string can be the representative string for more than one interval. However, if it is the representative string for more than two intervals, then it must span all except possibly the first and last of those intervals. Hence, it can be a split string for at most two intervals. Figure 3 shows examples of these cases.

The second idea is for the client to respond with shorter codewords for likely strings and longer codewords for less likely strings. However, the client does not have direct access to the distribution \( D \). Instead, each round, the server will send a good encoding of the strings, in addition to the strings themselves. The encoding we use is based on the number of intervals each representative string spans.

For a binary string \( t \), let \( 0 < t < 1 \) be the real number in \((0, 0.1 + 2^{-2})\) whose digits after the decimal place are the bits of \( t \). The following protocol considers intervals of the form \((0.1 + 2^{-j}, 0.1 + 2^{-2})\), where \( j \) is the length of the string \( t \). The server maintains an interval containing \( F(x) \) and, each iteration, the interval is shrunk until it is contained in \((F(x), F(x) + D(x))\).

Consider an execution of the above protocol in which \( m \) rounds occur. Let \( t_0 \) and \( j_0 \) denote the initial values of \( t \) and \( j \). Let \( t_i \) and \( j_i \) denote the values of \( t \) and \( j \), respectively, at the end of the \( i \)th round. Let \( u_i = 0.t_i \) and \( v_i = 0.t_i + 2^{-j_i} \).

**Theorem 2** Round-Efficient-Split has performance characteristics \([O(2^nH(D)), H(D) + 2, O(2^nH(D)), \frac{H(D)+1}{k} + 2]\).

**Proof.** Assume the client holds string \( x \) and let \( i = \lceil \log_2 \frac{1}{D(x)} \rceil \). The rounds of the protocol consist of some number of split rounds, in which the server receives codewords for split strings, followed by a final span round where the server receives a codeword for a span string.

At the end of each split round, the server reduces the current subinterval by a factor of \( 2^k \). If \( m = \lceil \frac{i+1}{k} \rceil \) split rounds occur, then \((u_m, v_m)\) has length at most \( 2^{-i} \). Since \( D(x) \geq 2^{-i} \) and \( F'(x) \in (u_m, v_m) \), if follows that \((u_m, v_m)\) lies completely within the interval \((F(x), F(x) + D(x))\). Thus, there is a span string for this interval. In the next round, the client sends the corresponding codeword to the server, and the server learns the string. Thus, the protocol performs at most \( \lceil \frac{i+1}{k} \rceil + 1 \) rounds. It follows that the expected number of rounds is:

\[
\sum_{x \in D} D(x) \left( \left\lceil \frac{\log_2 \frac{1}{D(x)}}{k} \right\rceil + 1 \right) \leq \sum_{x \in D} D(x) \left( \frac{\log_2 \frac{1}{D(x)}}{k} + 2 \right) = \frac{H(D)+1}{k} + 2.
\]
**Algorithm Round-Efficient-Split(\(k\))**

1. Initialize \(t\) to be the empty string. Let \(j\) be the length of \(t\), which is initially zero.

2. Repeat the following until there is only one string \(z\) such that \(F'(z) \in (0.t, 0.t + 2^{-j}]\):
   - Let \(a_1, \ldots, a_{2^k}\) be the \(2^k\) distinct binary strings of length \(k\). The server finds the representative strings for the intervals \((0.ta_i, 0.ta_i + 2^{-j-i}]\), \(i = 1, \ldots, 2^k\). Each span string \(y\) is assigned a weight, which is the number of these intervals for which \(y\) is a span string. Each split string is assigned weight 1.
   - The server then generates a Shannon code [5] for the set of representative strings, based on the weights of these strings. If a string is both a span string and a split string, two separate codewords are generated; otherwise, only one codeword is generated for that string.
   - First the set of span strings are sent to the client, in lexicographic order, each with its respective codeword. These are followed by the split strings, in lexicographical order, with their respective codewords.
   - If the client's string \(x\) is one of the span strings, then the corresponding codeword is sent back to the server. Otherwise, among the split strings it has received, the client finds the lexicographically least string \(y\) such that \(x \preceq y\). It sends back the codeword for \(y\).
   - The server does one of the following, depending on the codeword it receives from the client:
     - If the server receives the codeword for a span string, then it outputs the string and terminates.
     - Otherwise, the server receives the codeword for a split string, \(y\). Let \(a_l\) be the lexicographically least string such that \(y\) is not a span string for \((0.ta_l, 0.ta_l + 2^{-j-l}]\) and \(y\) is a \((0.ta_l + 2^{-j-l})\)-split string. The server appends \(a_l\) to \(t\) and increments \(j\) by \(k\), so that \(j\) is the new length of \(t\).

3. The server outputs \(z\).
Next, consider the number of bits sent by the client. Each round, the server sends a Shannon code to the client based on the weights of the different representative strings. Recall that each weight is a positive integer and the sum of these weights is at most $2^k$. It follows that no codeword has length greater than $k$.

If at most $m$ rounds occur, the client sends at most $km$ bits. Otherwise, there are $m$ split rounds, during which the clients sends a total of at most $km$ bits, followed by a final span round, in which the client sends the codeword for $x$. Let $e$ be the length of this codeword. Since a string of weight at least $2^{k-e+1}$ has length at most $e - 1$, the string $x$ has weight at most $2^{k-e+1} - 1$ and, hence, spans at most $2^{k-e+1} - 1$ intervals. However, $x$ can come arbitrarily close to spanning $2^{k-e+1} + 1$ subintervals, since it can nearly span the subintervals on either end. (See Figure 4.) Each subinterval has length $2^{-k(m+1)}$. Therefore,

$$D(x) < (2^{k-e+1} + 1)2^{-k(m+1)} < (2^{k-e+1} + 2^{k-e+1})2^{-k(m+1)} = 2^{2-e-km}$$

and the total number of bits sent by the client is at most

$$km + e < 2 - \log_2 D(x) = \log_2 \frac{1}{D(x)} + 2.$$
Thus, the expected number of bits sent by the client is less than
\[
\sum_{x \in D} D(x) \left( \log_2 \frac{1}{D(x)} + 2 \right) = H(D) + 2.
\]

During each round, the server sends as many as \(2^k\) split strings, each with a codeword of length at most \(k\) and a bit to indicate whether the codeword is for a span string or a split string. Thus, the total number of bits sent by the server is at most \((2^k(n + k + 1))(\frac{H(D) + 2}{k} + 1) = O(2^k n H(D))\).

At most \(n\) black box queries are used to find each representative string. Since the expected number of representative strings used by the protocol is \(O(2^k n H(D))\), the expected total number of black box queries is \(O(2^k n H(D))\).

\[\Box\]

3 Discussion

For the two protocols presented in this paper, the expected number of bits sent by the client is at most \(H(D) + 2\), an improvement over previous protocols.

Adler and Maggs [1] show that for any protocol and any entropy \(h\), there is a distribution \(D_h\) with \(H(D_h) = h\) such that the sum of the expected number of bits sent by the client and the server must be at least \(n\). Thus, for small \(h\), if the expected number of bits sent by the client is close to \(H(D_h)\), then the expected number of bits sent by the server is \(\Omega(n)\). In protocol Bit-Efficient-Split, the expected number of client bits sent is at most \(H(D) + 2\), which is within 2 of the optimal. However, this comes at the expense of an increase in the expected number of bits sent by the server, to a factor of \(H(D)\) over the optimal. For the protocol Round-Efficient-Split(k), the expected number of bits sent by the server gets even larger as \(k\) increases and the number of rounds decreases. It would be valuable to develop a protocol where the expected number of bits sent by the client is \(H(D) + O(1)\) and the expected number of bits sent by the server is \(O(n)\), or prove lower bound showing that this is not possible.

References


**John Watkinson** is with Modelwire Inc., New York, NY 10012 USA, john@modelwire.com.

**Micah Adler** is with the Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, 01003-4610, USA, micah@cs.umass.edu.

**Faith E. Fich** is with the Department of Computer Science, University of Toronto, Toronto, Ontario, CANADA M5S 3G4, fich@cs.toronto.edu.
On Finding Minimum Deadly Sets for Directed Networks

NORBERT ZEH
Carleton University, Canada

NICOLA SANTORO
Carleton University, Canada

Abstract
Given a set $S$ of elements in a directed network that are initially faulty, an element becomes (functionally) faulty if all its in-neighbors or all its out-neighbors are (functionally) faulty. A set $S$ of initially faulty elements is called deadly if it causes the entire network to become faulty according to the above rule. We show that finding a minimum deadly set is NP-hard for arbitrary directed networks. For directed acyclic graphs (DAGs), we show that finding a weighted minimum deadly set is no harder than finding a minimum cut. We also study the case where a vertex becomes faulty if at least a certain percentage of its in-neighbors or out-neighbors is faulty. We call a set $S$ of initially faulty elements $\varepsilon$-deadly if it causes the whole network to become faulty using this $\varepsilon$-majority rule. We show that finding a minimum $\varepsilon$-deadly set is NP-hard even for a restricted subclass of directed acyclic graphs.

Keywords
distributed computing, fault tolerance, majority rule, NP-hardness, minimum cut

1 Introduction

Background In most communication-based systems, the behavior of a single entity depends on the status of its neighboring entities in the system. This is true by definition in systems such as cellular automata, neural networks, or coupled map lattices; such dependencies can be found, sometimes unexpectedly, at different levels in VLSI systems, data communication networks, and distributed systems. This “locality” property can be used as a basic construct for distributed computations [24] and has been extensively employed in a variety of applications. Thus, it regulates the behavior of many communication protocols.

In all these environments, faulty elements can induce a faulty behavior in their neighbors. This is for example the case when the elements of a network maintain the consistency of crucial data by comparing their local copies with the ones
held by their neighbors, and resolving inconsistencies by performing majority voting [22]; thus, if the majority of its neighbors has corrupted data, a non-faulty element will exhibit a faulty behavior (e.g., its data will become corrupted) and will therefore be indistinguishable from a faulty one.

A large class of problems related to security and fault-tolerance have been modeled, analyzed and solved using this view of the system. In these studies, the system is viewed as a graph; vertices correspond to system entities; edges describe the neighborhood relation (e.g., direct communication links between entities). Every vertex is initially colored either “black” (faulty) or “white” (non-faulty). At each local time step, it recolors itself according to the colors held by the majority of its neighbors. Depending on the initial assignment of colors to the nodes and the definition of majority, different dynamics emerge.

These dynamics have been extensively studied in the context of synchronous systems (mostly cellular automata) with different majority functions (simple majority, strong majority, weighted threshold functions, convex functions) and different coloring sets (e.g., see [1, 11, 16, 25]).

In the context of distributed computing, the interest has been in simple and strong majority; the focus has been on the patterns of initial faults which may cause the entire system to behave in a faulty manner. In terms of system dynamics, these are the patterns for which the system converges to a monochromatic fixed point. They have been called dynamos by Peleg who introduced their study in [23]. Most of the results are known for the static version of this process, that is, considering only one or two steps in the evolution [3, 4, 14, 22]. In the dynamic case, results on dynamos exist for a variety of topologies, including chordal rings, tori, and most interconnection networks [7, 8, 9, 10, 15, 23]. Recently nondeterministic rules and weighted majority functions have been investigated in [12, 13, 17].

All these studies have almost exclusively considered undirected networks. In this paper we are interested in the dynamics of majority rules in the more general case of directed networks. Surprisingly little is known for these systems. The only relevant results come from a separate line of investigation carried out in the area of VLSI arrays.

In these systems, fault-tolerance is commonly achieved by the combined use of component redundancy, bypass links, and reconfiguration techniques. However, a small number of strategically located faults can render the entire array unusable, regardless of the amount of redundancy and the cleverness of the reconfiguration technique; these sets of faults were called deadly and the patterns they formed were termed catastrophic fault patterns (CFP) by Nayak et al., who introduced their study in [20]. Clearly, a deadly set leads the system to a monochromatic fixed point, as does a dynamo. The fundamental difference between the dynamo and the CFP setting is that, in the latter, links are unidirectional and an

\[1\] That is, majority is defined as half of the neighbors. The difference is how ties are broken.
element becomes “dead” only if all its in-neighbors (out-neighbors) are “dead”. 2

There have been several investigations on catastrophic fault patterns and their properties, the complexity of recognizing whether a set of faults is deadly, the efficient generation of such CFP, and how to route if the pattern is not catastrophic, etc. [27, 26, 18, 19, 20, 21].

Unidirectionality makes the results on CFP directly relevant for our investigation. However, those studies were limited to a restricted class of networks; indeed, due to the nature of the application system (VLSI array with regular bypass links), they apply only to directed chordal rings in one and two dimensions.

Our Results In this paper, we study the complexity of finding deadly and \( \varepsilon \)-deadly sets for arbitrary directed networks. In Section 2, we introduce the most important terminology and prove some basic properties of deadly and \( \varepsilon \)-deadly sets, which are the tools employed to prove the results of Sections 3, 4, and 5. In Section 3, we show that finding a minimum deadly set in the sense of [20] for arbitrary directed graphs is NP-hard, while Section 4 provides a linear time reduction of the problem of finding a minimum deadly set in directed acyclic graphs to that of finding a minimum cut in an st-graph constructed from the given DAG. In Section 5, we prove that finding a minimum \( \varepsilon \)-deadly set is NP-hard even for a very restricted class of directed acyclic graphs.

Definitions A directed graph (directed network) \( G = (V, E) \) is an ordered pair of sets \( V \) and \( E \), where the elements in \( E \) are ordered pairs \((v, w)\) of elements \( v, w \in V \). We call the elements of \( V \) the vertices or nodes of \( G \); the elements of \( E \) are the edges of \( G \). For an edge \((v, w) \in E\), we call vertices \( v \) and \( w \) the endpoints of \((v, w)\); \( v \) and \( w \) are adjacent. The in-neighborhood of a vertex \( v \in V \) is the set \( N^-(v) = \{ u \in V : (u, v) \in E \} \). Analogously, the out-neighborhood of a vertex \( v \in V \) is the set \( N^+(v) = \{ w \in V : (v, w) \in E \} \). The in-degree and out-degree of a vertex \( v \) are defined as \( \deg^-(v) = |N^-(v)| \) and \( \deg^+(v) = |N^+(v)| \), respectively. A path in \( G \) is a sequence \( P = (v_0, v_1, \ldots, v_k) \) of vertices in \( G \) such that \((v_{i-1}, v_i) \in E\), for all \( 1 \leq i \leq k \). \( P \) is a cycle if \( v_0 = v_k \). A directed acyclic graph (DAG) is a directed graph that does not contain cycles. A vertex \( v \) in a DAG \( G = (V, E) \) is a source if \( \deg^-(v) = 0 \); vertex \( v \) is a sink if \( \deg^+(v) = 0 \). An st-graph is a DAG \( G = (V, E) \) with exactly one source \( s \) and exactly one sink \( t \). In subsequent sections, we frequently perform the operation \( G \cap S \) for a graph \( G = (V, E) \) and a vertex set \( S \). Viewing \( G \) as a collection of vertices and edges, this operation is naturally defined as \( G \cap S = V \cap S \).

---

2 The initially faulty elements are dead by definition; we are using the original terminology of [20] here.
2 Deadly Sets, Propagation Sequences, and Immortal Subgraphs

In this section, we study propagation sequences and immortal subgraphs, which are the basic tools employed to prove the results in Sections 3, 4, and 5. Given an assignment \( \omega : V \to \mathbb{R}^+ \) of weights to the vertices of a graph \( G = (V, E) \), the weight \( \omega(S) \) of a subsets \( S \subseteq V \) is defined as \( \omega(S) = \sum_{v \in S} \omega(v) \). A subset \( S \subseteq V \) of vertices of \( G \) induces an \( \epsilon \)-propagation sequence \( S = S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots \) of subsets of \( V \), for \( 0 < \epsilon \leq 1 \), where a vertex \( v \in V \) is contained in set \( S_i \), \( i > 0 \), if and only if

1. \( v \in S_{i-1} \),
2. \( \mathcal{N}^-(v) \neq \emptyset \) and \( \omega(S_{i-1} \cap \mathcal{N}^-(v)) \geq \epsilon \cdot \omega(\mathcal{N}^-(v)) \), or
3. \( \mathcal{N}^+(v) \neq \emptyset \) and \( \omega(S_{i-1} \cap \mathcal{N}^+(v)) \geq \epsilon \cdot \omega(\mathcal{N}^+(v)) \).

Set \( S \) is \( \epsilon \)-deadly if there exists some \( k \geq 0 \) such that \( S_i = V \), for all \( i \geq k \). We call a subgraph \( I \) of \( G \) \( \epsilon \)-immortal if \( I \cap S = \emptyset \) implies that \( I \cap S_i = \emptyset \), for all \( i \geq 0 \). We call a set \( S \) deadly if it is 1-deadly. A subgraph \( I \) of \( G \) is immortal if it is 1-immortal. The weighted or unweighted (\( \epsilon \)-deadly set problem is the problem of finding a minimum \( (\epsilon \)-deadly set for a weighted or unweighted directed graph.

Next we provide necessary and sufficient conditions for a subgraph of \( G \) to be \( \epsilon \)-immortal, and for a subset of vertices of \( G \) to be \( \epsilon \)-deadly.

**Lemma 1** A subgraph \( I \) of \( G \) is \( \epsilon \)-immortal, \( 0 < \epsilon \leq 1 \), if and only if every vertex \( v \in I \) satisfies the following two conditions:

1. Either \( \mathcal{N}^-(v) = \emptyset \) or \( \omega(\mathcal{N}^-(v) \cap I) > (1 - \epsilon)\omega(\mathcal{N}^-(v)) \),
2. Either \( \mathcal{N}^+(v) = \emptyset \) or \( \omega(\mathcal{N}^+(v) \cap I) > (1 - \epsilon)\omega(\mathcal{N}^+(v)) \).

**Proof.** First assume that every vertex \( v \in I \) satisfies the above conditions and that \( S \cap I = \emptyset \). We have to show that \( S_i \cap I = \emptyset \), for all \( i \geq 0 \), where \( S = S_0, S_1, S_2, \ldots \) is the \( \epsilon \)-propagation sequence induced by \( S \). The proof is by induction.

The base case is trivial because \( S = S_0 \) and \( S \cap I = \emptyset \). So assume that \( S_i \cap I = \emptyset \), for \( 0 \leq i \leq k \). We have to show that \( S_{k+1} \cap I = \emptyset \). Assume for the sake of contradiction that there is a vertex \( v \in S_{k+1} \cap I \). Note that \( v \notin S_k \). Thus, either \( \mathcal{N}^-(v) \neq \emptyset \) and \( \omega(\mathcal{N}^-(v) \cap S_k) \geq \epsilon \cdot \omega(\mathcal{N}^-(v)) \), or \( \mathcal{N}^+(v) \neq \emptyset \) and \( \omega(\mathcal{N}^+(v) \cap S_k) \geq \epsilon \cdot \omega(\mathcal{N}^+(v)) \). However, \( \mathcal{N}^-(v) \neq \emptyset \) implies that \( \omega(\mathcal{N}^-(v) \cap I) > (1 - \epsilon)\omega(\mathcal{N}^-(v)) \), by the first condition of the lemma, and \( \mathcal{N}^+(v) \neq \emptyset \) implies that \( \omega(\mathcal{N}^+(v) \cap I) > (1 - \epsilon)\omega(\mathcal{N}^+(v)) \), by the second condition of the lemma. This leads to a contradiction because \( S_k \cap I = \emptyset \) and thus \( \omega(\mathcal{N}^-(v) \cap (I \cup S_k)) > \omega(\mathcal{N}^-(v)) \) and \( \omega(\mathcal{N}^+(v) \cap (I \cup S_k)) > \omega(\mathcal{N}^+(v)) \).

---

3In an unweighted graph, all vertices have equal weight.
Now assume that \( I \) is \( \varepsilon \)-immortal. We have to show that every vertex in \( I \) satisfies the two conditions of the lemma. So let \( S = G \setminus I \) and \( S = S_0, S_1, S_2, \ldots \) be the \( \varepsilon \)-propagation sequence induced by \( S \). Let \( v \) be a vertex of \( I \) that does not satisfy the first condition of the lemma. That is, \( \mathcal{N}^- (v) \neq \emptyset \) and \( \omega(\mathcal{N}^- (v) \cap I) \leq (1 - \varepsilon) \omega(\mathcal{N}^- (v)) \). Then \( \omega(\mathcal{N}^- (v) \cap S) \geq \varepsilon \cdot \omega(\mathcal{N}^- (v)) \) because \( S \cup I = V \), so that \( v \in S_1 \), contradicting the \( \varepsilon \)-immortality of \( I \). A similar argument applies if \( v \) does not satisfy the second condition.

\[ \square \]

**Lemma 2** A subset \( S \) of the vertices of \( G \) is \( \varepsilon \)-deadly, \( 0 < \varepsilon \leq 1 \), if and only if \( S \cap I \neq \emptyset \), for every \( \varepsilon \)-immortal subgraph \( I \) of \( G \).

**Proof.** First assume that there is an \( \varepsilon \)-immortal subgraph \( I \) of \( G \) such that \( I \cap S = \emptyset \). Let \( S = S_0, S_1, S_2, \ldots \) be the \( \varepsilon \)-propagation sequence induced by \( S \). By the definition of \( \varepsilon \)-immortal subgraphs, \( I \cap S_i = \emptyset \), for \( i \geq 0 \). Thus, \( S \) cannot be \( \varepsilon \)-deadly.

Now assume that \( S \) is not \( \varepsilon \)-deadly. We show that there has to be an \( \varepsilon \)-immortal subgraph \( I \) of \( G \) such that \( I \cap S = \emptyset \). As \( S \) is not \( \varepsilon \)-deadly, there has to be an index \( k \) such that \( S_i = S_k \neq V \), for \( i \geq k \). Let \( v_0 \in V \setminus S_{k+1} \).

We construct an \( \varepsilon \)-immortal subgraph \( I \) of \( G \) such that \( I \cap S = \emptyset \). We do this by constructing a sequence of subgraphs \( I_0 \subset I_1 \subset \cdots \subset I_r \) such that \( V(I_0) = \{ v_0 \} \) and \( I_r = I \). Given that we have constructed a sequence \( I_0, \ldots, I_i \), we show that either all vertices in \( I_j \) satisfy the conditions of Lemma 1, or we can add a vertex not in \( S_{k+1} \) to \( I_j \) to obtain a proper supergraph \( I_{j+1} \) of \( I_j \) with \( I_{j+1} \cap S_{k+1} = \emptyset \). As we can repeat this augmentation process only a finite number of times, we must finally obtain an \( \varepsilon \)-immortal subgraph \( I_r \) of \( G \) with \( I_r \cap S_{k+1} = \emptyset \). As \( S \subseteq S_{k+1} \), \( I_r \cap S = \emptyset \), as desired.

So assume that there is a vertex \( v \in I_j \) which does not satisfy the first condition of Lemma 1. That is, \( \mathcal{N}^- (v) \neq \emptyset \) and \( \omega(\mathcal{N}^- (v) \cap I_j) \leq (1 - \varepsilon) \omega(\mathcal{N}^- (v)) \). As \( v \notin S_{k+1} \), \( \omega(\mathcal{N}^- (v) \cap S_k) \geq \varepsilon \cdot \omega(\mathcal{N}^- (v)) \). Thus, \( \omega(\mathcal{N}^- (v) \cap I_j) + \omega(\mathcal{N}^- (v) \cap S_k) < \omega(\mathcal{N}^- (v)) \), and there has to be a vertex \( u \in \mathcal{N}^- (v) \setminus (I_j \cup S_k) \). Since \( S_k = S_{k+1} \), \( u \notin I_j \cup S_{k+1} \). We obtain \( I_{j+1} \) by adding \( u \) to \( I_j \). If vertex \( v \) does not satisfy the second condition of Lemma 1, a similar procedure finds a vertex \( w \in \mathcal{N}^+ (v) \setminus (I_j \cup S_{k+1}) \) to be added to \( I_j \).

\[ \square \]

### 3 Deadly Sets for Arbitrary Directed Graphs

In order to prove the NP-hardness of all variants of the deadly set problem for arbitrary directed graphs, we provide a linear time reduction of the vertex cover problem to the unweighted deadly set problem. A vertex cover of an undirected graph \( G \) is a set \( C \) of vertices of \( G \) such that every edge has at least one endpoint in \( C \). It has been shown that finding such a set \( C \) of minimum cardinality is NP-hard (e.g., see [6]).
Theorem 1 The unweighted and weighted deadly and \( \varepsilon \)-deadly set problems, \( 0 < \varepsilon < 1 \), are NP-hard on directed graphs.

Proof. Given an undirected graph \( G \), we construct a directed graph \( G' = (V, E') \) with edge set \( E' = \{(v, w), (w, v) : \{v, w\} \in E\} \). Every edge in \( G \) corresponds to a cycle in \( G' \). Thus, every deadly set for \( G' \) covers every edge in \( G \) because cycles are immortal. Conversely, if a vertex cover does not contain a vertex \( v \) of \( G \), all neighbors of \( v \) must be in the vertex cover. Thus, every vertex cover for \( G \) is deadly for \( G' \).

4 Deadly Sets for Directed Acyclic Graphs

Next we provide a linear time reduction of the problem of finding a minimum deadly set for a given DAG to that of finding a minimum cut in an \( st \)-graph constructed from the given DAG. A cut of an \( st \)-graph \( G = (V, E) \) is a set \( C \subseteq E \) of edges such that any path from \( s \) to \( t \) contains at least one edge in \( C \). As such a minimum cut can be found in \( O(nm \log(n^2/m)) \) time [2], this provides an efficient solution for the deadly set problem on DAGs.

Theorem 2 It takes \( O(MC(n, m)) \) time to solve the weighted deadly set problem on directed acyclic graphs, where \( MC(n, m) \) is the time required to compute a minimum cut in an \( st \)-graph with \( n \) vertices and \( m \) edges.

Proof. Let a DAG \( G = (V, E) \) and a function \( \omega : V \to \mathbb{R}^+ \) assigning weights to the vertices of \( G \) be given. Assume that \( G \) does not contain isolated vertices. We show how to deal with the general case at the end of the proof. We construct an \( st \)-graph \( G' = (V', E') \) with vertex set \( V' = (V \times \{0, 1\}) \cup \{s, t\} \) and edge set

\[
E' = \{(s, (v, 0)) : v \text{ is a source in } G\} \cup \\
\{(w, 1, t) : w \text{ is a sink in } G\} \cup \\
\{(v, 0, (v, 1)) : v \in V\} \cup \\
\{((v, 1), (w, 0)) : (v, w) \in E\}.
\]

(See Figure 1.) We define a function \( \omega' : E' \to \mathbb{R}^+ \) assigning weights to the edges of \( G' \) as

\[
\omega'(e) = \begin{cases} 
\omega(v) & \text{if } e = (s, (v, 0)) \\
\omega(w) & \text{if } e = ((w, 1), t) \\
\omega(v) & \text{if } e = ((w, 0), (v, 1)) \\
\max(\omega(v), \omega(w)) & \text{if } e = ((v, 1), (w, 0))
\end{cases}
\]

We show how to construct a deadly set \( S \) of \( G \) from a cut \( C \) of \( G' \) and that \( S \) is a minimum deadly set of \( G \) if an only if \( C \) is a minimum cut for \( G' \).
Given a cut $C$ of $G'$, we construct a deadly set $S$ for $G$ as follows: For every edge $e' = (v', w') \in C$, we add $v$ to $S$ if $v' = (v, x)$ for some $v \in V$ and $x \in \{0, 1\}$. Otherwise, $e' = (s, (w, 0))$, and we add $w$ to $S$.

First we prove that $S$ is deadly for $G$. Assume the contrary. That is, there is an immortal subgraph $I$ of $G$ such that $I \cap S = \emptyset$. A simple inductive argument shows that $I$ must contain a source-to-sink path $P = (v_0, \ldots, v_k)$ as a subgraph. Then the corresponding st-path $P' = (s, (v_0, 0), (v_0, 1), (v_1, 0), (v_1, 1), \ldots, (v_k, 0), (v_k, 1), t)$ in $G'$ cannot contain an edge in $C$, a contradiction.

It is easy to see that $w(S) \leq w(C)$. Thus, the weight of a minimum deadly set of $G$ does not exceed the weight of a minimum cut in $G'$. Now assume that $C$ is a minimum cut of $G'$, and there exists a minimum deadly set $S'$ of $G$ such that $w(S') < w(C)$. Let $C'$ be the edge set $\{(v, 0), (v, 1) : v \in S'\}$. Thus, $w(C') = w(S') < w(C)$. We show that $C'$ is a cut for $G'$, thereby contradicting the minimality of $C$.

So assume that there exists an st-path $(s, (v_0, 0), (v_0, 1), (v_1, 0), (v_1, 1), \ldots, (v_k, 0), (v_k, 1), t)$ in $G'$ that does not contain a single edge in $C'$. Then the corresponding source-to-sink path $P' = (v_0, v_1, \ldots, v_k)$ in $G$ cannot contain a vertex in $S'$. But this is a contradiction, as $P'$ is immortal.

If $G$ contains isolated vertices, let $H$ be the subgraph of $G$ obtained by removing all isolated vertices. By Lemmas 1 and 2, every isolated vertex has to be contained in any deadly set for $G$. Thus, it is sufficient to find a deadly set for $H$. Let $n' \leq n$ be the number of vertices in $H$, and $m' \leq m$ be the number of edges in $H$. Then $m' \geq n'/2$. The graph $H'$ constructed from $H$ by the above procedure has $2n' + 2 = O(n)$ vertices and at most $3n' + m' = O(m)$ edges. Thus, finding a minimum deadly set for $G$ takes $O(n + m) + MC(O(n), O(m)) = O(MC(n, m))$ time. \qed
5 $\epsilon$-Deadly Sets for Directed Acyclic Graphs

We now show that the $\epsilon$-deadly set problem is NP-hard even for a restricted subclass of directed acyclic graphs whose vertices have weights within a restricted range.

**Theorem 3** The weighted $\epsilon$-deadly set problem, $0 < \epsilon < 1$, is NP-hard for directed acyclic graphs whose nodes have in-degree and out-degree bounded by some constant $D$ and weights between $w$ and $W$, provided that $D \geq 2 + \max\left(\left\lceil \frac{2w}{\epsilon W}\right\rceil, \left\lceil \frac{\epsilon w}{(1-\epsilon)W}\right\rceil\right)$.

We prove Theorem 3 by constructing a graph $G(F, \epsilon)$ for a given Boolean formula $F$ in 3-CNF such that $F$ is satisfiable if and only if $G(F, \epsilon)$ has an $\epsilon$-deadly set of low weight. We show that $G(F, \epsilon)$ can be constructed in time polynomial in the size of $F$ and ensure that it satisfies the conditions of Theorem 3.

We need the following definitions and results. Given a set $X$ of Boolean variables, a Boolean formula $F(x_1, \ldots, x_n)$ is in conjunctive normal form with clauses of size exactly 3 (3-CNF) if

1. $F(x_1, \ldots, x_n) = C_1 \land C_2 \land \cdots \land C_m$,
2. $C_i = \lambda_{i,1} \lor \lambda_{i,2} \lor \lambda_{i,3}$, for $1 \leq i \leq m$, and
3. For all pairs $(i, j)$, $1 \leq i \leq m$, $1 \leq j \leq 3$, there exists a $k$, $1 \leq k \leq n$, such that $\lambda_{i,j} = x_k$ or $\lambda_{i,j} = \bar{x}_k$.

A Boolean formula $F(x_1, \ldots, x_n)$ is satisfiable if there is a truth assignment $\beta : X \to \{\text{true, false}\}$ such that $F(\beta(x_1), \ldots, \beta(x_n)) = \text{true}$. It has been shown that the problem of deciding whether a given formula $F$ in 3-CNF (3-SAT) is satisfiable is NP-hard (e.g., see [5]).

Given formula $F$, the graph $G(F, \epsilon)$ consists of a number of subgraphs, called gadgets. Most of these gadgets are duplicated, so that we can think of $G(F, \epsilon)$ as being composed of $2n$ identical layers $L_1, \ldots, L_k$ (see Figure 2). Each of these layers is comprised of $2n$ multiplier gadgets, $n$ negator gadgets, $m$ clause gadgets, one final gadget, and one feedback gadget. We add $2n$ literal gadgets to $G(F, \epsilon)$, which are connected to all layers of $G(F, \epsilon)$. We describe $G(F, \epsilon)$ for the case $\epsilon \leq \frac{1}{2}$.

For $\epsilon > \frac{1}{2}$, $G(F, \epsilon)$ contains an additional booster gadget. Details will appear in the full paper.

We first describe the gadgets in one layer $L_i$ and show how they are connected to form layer $L_i$. The other layers are identical. Then we describe the literal gadgets and show how they are linked to the layers. Assume that we have normalized $w$ and $W$ so that $w = 1$. All nodes in $G(F, \epsilon)$ have weight 1 unless stated otherwise.
There is one multiplier gadget in $L_i$ for each of the $2n$ literals $x_1, \tilde{x}_1, x_2, \tilde{x}_2, \ldots, x_n, \tilde{x}_n$ over $X$ (see Figure 3a). The multiplier gadget for literal $\lambda$ is a binary tree $\Lambda$ with root $\bar{\lambda}$. The edges of $\Lambda$ are directed from the root towards the leaves. There is one more leaf in $\Lambda$ than clauses in $F$ containing $\lambda$. We augment $\Lambda$ with additional nodes. In particular, every internal node of $\Lambda$ has $\left\lfloor \frac{W}{\epsilon} \right\rfloor$ additional leaves of weight $W$ as children. These extra children serve as feedback blockers and are not considered regular leaves of $\Lambda$.

There is one negator gadget in $L_i$ for each of the $n$ variables in $X$ (see Figure 3b). The negator gadget for variable $x_j$ consists of a single negator node $\chi_{i,j}$.

There is one clause gadget per clause $C_j$ in $F$ (see Figure 3c). Each such clause gadget consists of two nodes $\gamma_{i,j}$ and $\bar{\gamma}_{i,j}$ and an edge from $\bar{\gamma}_{i,j}$ to $\gamma_{i,j}$. Node $\gamma_{i,j}$ is called the clause node for clause $C_j$ in layer $L_i$.

There is one final gadget in $L_i$ (see Figure 3f). It consists of a binary tree $\Psi_i$ with root $\psi_i$, $k = \left\lceil \frac{k}{1+\epsilon} \right\rceil$ nodes $\psi_{i,1}, \ldots, \psi_{i,k}$ of weight $W$ each, and $k$ final nodes $\psi_{i,1}, \ldots, \psi_{i,k}$ of weight $W$ each. We add edges from $\psi_i$ to all nodes $\psi_{i,j}$, $1 \leq j \leq k$. There is one leaf in $\Psi_i$ for every clause in $F$ and every variable in $X$, $n + m$ in total. The edges in $\Psi_i$ are directed from the leaves towards the root. Every internal node of $\Psi_i$ has a number of additional children; the purpose of these children is to increase the in-weight of every regular node in $\Psi_i$, in order to enforce a logical AND behavior of the nodes in $\Psi_i$. These additional children are not considered regular leaves of $\Psi_i$. Their number depends on $W$ and $\epsilon$. In particular, if $W \leq \frac{1}{\epsilon}$, every node has $\left\lfloor \frac{2W}{\epsilon^2} \right\rfloor$ additional children of weight $W$ each. If $\frac{1}{\epsilon} < W < \frac{2W}{\epsilon}$, every node in $\Psi_i$ has 2 additional children of weight $\frac{W}{\epsilon}$. If $W \geq \frac{2W}{\epsilon}$, every node in $\Psi_i$ has one extra child of weight $\frac{2W}{\epsilon}$.

There is one feedback gadget in $L_i$ (see Figure 3d). It consists of a binary tree $\Phi_i$ with root $\phi_i$ and a feedback node $\phi_i$. There is an edge from $\phi_i$ to $\phi_i$. Tree $\Phi_i$ has one leaf for each of the $2n$ literals over $X$. The edges in $\Phi_i$ are directed from the root towards the leaves. Every internal node of $\Phi_i$ has $\left\lfloor \frac{W}{\epsilon^2} \right\rfloor$ additional children of weight $W$ that serve as feedback blockers and are not considered to be regular leaves of $\Phi_i$.

We connect these gadgets to form layer $L_i$. We add edges from the leaves in the multiplier gadgets to the nodes in the clause gadgets as follows: Consider a clause gadget corresponding to clause $C_j = \lambda_{j,1} \lor \lambda_{j,2} \lor \lambda_{j,3}$. Then we add edges from two leaves in trees $\Lambda_{i,j,1}$ and $\Lambda_{i,j,2}$ to $\gamma_{i,j}$ and an edge from a leaf in tree $\Lambda_{i,j,3}$ to $\gamma_{i,j}$. Every leaf in each tree $\Lambda_i$ is connected to at most one node in a clause gadget. This leaves one regular leaf per tree $\Lambda_i$ with out-degree 0. We add edges from the remaining two leaves in trees $X_{i,j}$ and $\tilde{X}_{i,j}$ to the negator node $\chi_{i,j}$, for every variable $x_j \in X$. We add edges from the clause nodes in all clause gadgets and the negator nodes in all negator gadgets to the leaves of the final gadget, so that every leaf in the final gadget is connected to exactly one node in a clause or negator gadget, and vice versa. We add edges from the feedback node $\phi_i$ to all final nodes $\Psi_{i,1}, \ldots, \Psi_{i,k}$.
Figure 3: The gadgets in $G(F, \varepsilon)$. Regular nodes are white. Additional nodes such as feedback blockers are black. (a) A multiplier gadget. (b) A negator gadget. (c) A clause gadget. (d) A feedback gadget. (e) A literal gadget. (f) A final gadget.

Apart from layers $L_1, \ldots, L_k$, graph $G(F, \varepsilon)$ contains a number of literal gadgets. There is one literal gadget for each of the $2n$ literals over the set $X$ of variables in $F$ (see Figure 3e). The literal gadget for literal $\lambda$ consists of two binary trees $\Delta$ and $\Gamma$ sharing the same root $\lambda$. The edges in $\Delta$ are directed from the leaves towards the root; the edges in $\Gamma$ are directed from the root towards the leaves. Both trees have $2n$ leaves. Let $\mu_1$ and $\mu_2$ be the children of $\lambda$ in $\Delta$, and $v_1$ and $v_2$ be the children of $\lambda$ in $\Gamma$. Then we guarantee that each of $\mu_1, \mu_2, v_1$, and $v_2$ has $n$ leaves as descendants. Let the descendant leaves of $\mu_1$ and $v_1$ be numbered $1$ through $n$, and the descendant leaves of $\mu_2$ and $v_2$ be numbered $n + 1$ through $2n$. The internal nodes in $\Gamma$ have $\lceil \frac{W}{n} \rceil$ additional children of weight $W$ each. The internal nodes in $\Delta$ have the same number of additional children as the internal nodes in tree $\Psi_i$ in a final gadget. Again, these additional children are not considered to be regular nodes of $\Delta$ and $\Gamma$.

To finish the construction of $G(F, \varepsilon)$, we have to connect the literal gadgets with layers $L_1, \ldots, L_k$. To do that we add edges between the leaves in all feedback gadgets and the leaves in trees $\Delta$, and between the leaves in trees $\Gamma$ and the roots of the trees in all multiplier gadgets. More precisely, if $\lambda = x_j$, we connect the leaf corresponding to $\lambda$ in tree $\Phi_i$ to the $i$-th leaf of tree $\Delta$, and the $i$-th leaf of tree $\Gamma$ to node $\lambda_i$; if $\lambda = \bar{x}_j$, we connect the leaf corresponding to $\lambda$ in tree $\Phi_i$ to the $(\lfloor i + n/2 \rfloor \mod 2n)$-th leaf of tree $\Delta$, and the $(\lfloor i + n/2 \rfloor \mod 2n)$-th leaf of tree $\Gamma$ to node $\lambda_i$. 

To illustrate this construction at an example, Figure 4 shows one layer of the graph $G(F, \varepsilon)$ constructed for the formula $F = (x_1 \lor x_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor x_4) \land (x_1 \lor x_3 \lor \overline{x}_4) \land (x_2 \lor x_3 \lor x_4)$ and $\varepsilon \leq \frac{1}{7}$. The connections of nodes in $L_i$ to literal gadgets are shown as dashed lines. Feedback, multiplier, clause, and final gadgets are shaded.

Figure 4: One layer $L_i$ of graph $G(F, \varepsilon)$, for $F = (x_1 \lor x_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor x_4) \land (x_1 \lor x_3 \lor \overline{x}_4) \land (x_2 \lor x_3 \lor x_4)$ and $\varepsilon \leq \frac{1}{7}$. The connections of nodes in $L_i$ to literal gadgets are shown as dashed lines. Feedback, multiplier, clause, and final gadgets are shaded.

Lemma 3 Every $\varepsilon$-deadly set of $G(F, \varepsilon)$ of weight at most $n$ contains exactly one of the nodes $x_j$ and $\overline{x}_j$, for all $x_j \in X$.

Proof. Assume that there exists an $\varepsilon$-deadly set $S$ of weight at most $n$ that does not satisfy the condition of the lemma. We show that there is an $\varepsilon$-immortal subgraph $I$ with $I \cap S = \emptyset$, thereby contradicting the $\varepsilon$-deadliness of $S$. 


First assume that there exists a variable $x_j \in X$ such that none of the literal gadgets for $x_j$ and $\bar{x}_j$ contains a node in $S$. As there are $2n$ layers, and no node has weight less than 1, there has to be a layer $L_i$ such that $L_i \cap S = \emptyset$. Let $H$ be the subgraph of $G(F, \varepsilon)$ consisting of the paths from $\phi_i$ to $x_j$ and $\bar{x}_j$, the paths from $x_j$ and $\bar{x}_j$ to $\chi_{i,j}$, and the paths from $\chi_{i,j}$ to all final nodes $\psi_{i,h}$ in $L_i$. Let $I$ be the graph induced by all nodes in $H$ and all sinks and sources adjacent to nodes in $H$. Then it is easy to verify that $I$ satisfies the conditions of Lemma 1 and that $I \cap S = \emptyset$. Hence, by Lemma 2, $S$ cannot be $\varepsilon$-deadly.

Thus, $S$ contains exactly one node in the literal gadgets for $x_j$ and $\bar{x}_j$, for every variable $x_j \in X$. Now assume that there exists a variable $x_j$ such that $x_j, \bar{x}_j \notin S$. Then there has to be a layer $L_i$ such that the paths from $\phi_i$ to $x_j$ and $\bar{x}_j$ and from $x_j$ and $\bar{x}_j$ to $\chi_{i,j}$ do not contain a node in $S$. Also, $L_i \cap S = \emptyset$. Thus, $I \cap S = \emptyset$, for the same subgraph $I$ as constructed above, and $S$ cannot be $\varepsilon$-deadly.

Lemma 3 implies that $G(F, \varepsilon)$ does not have an $\varepsilon$-deadly set of weight less than $n$. It also implies that every $\varepsilon$-deadly set $S$ of $G(F, \varepsilon)$ of weight $n$ corresponds to an assignment of truth values to the variables in $X$. In particular, we define $\beta(x_j) = \text{true}$ if $x_j \in S$, and $\beta(x_j) = \text{false}$ if $\bar{x}_j \in S$. Conversely, every assignment $\beta : X \rightarrow \{\text{true, false}\}$ of truth values to the variables in $X$ corresponds to a set $S_{\beta} = \{x_j : \beta(x_j) = \text{true}\} \cup \{\bar{x}_j : \beta(x_j) = \text{false}\}$.

**Lemma 4** The set $S_{\beta}$ is $\varepsilon$-deadly for $G(F, \varepsilon)$ if and only if $F(\beta(x_1), \ldots, \beta(x_n)) = \text{true}$.

**Proof.** It is easy to verify that the set $S_{\beta}$ is $\varepsilon$-deadly for graph $G(F, \varepsilon)$ if $F(\beta(x_1), \ldots, \beta(x_n)) = \text{true}$. So assume that $F(\beta(x_1), \ldots, \beta(x_n)) = \text{false}$. Then there has to be a clause $C_j = \lambda_{j,1} \lor \lambda_{j,2} \lor \lambda_{j,3}$ such that $\beta(C_j) = \beta(\lambda_{j,1}) \lor \beta(\lambda_{j,2}) \lor \beta(\lambda_{j,3}) = \text{false}$. Let $H$ be the subgraph of $G(F, \varepsilon)$ induced by the paths from $\phi_i$ to nodes $\lambda_{j,1}$, $\lambda_{j,2}$, and $\lambda_{j,3}$, the paths from $\lambda_{j,1}$, $\lambda_{j,2}$, $\lambda_{j,3}$ to $\gamma_{i,j}$, and the paths from $\gamma_{i,j}$ to all final nodes $\psi_{i,h}$ in layer $L_i$. Let $I$ be the subgraph of $G(F, \varepsilon)$ induced by all nodes in $H$ as well as all sources and sinks adjacent to vertices in $H$. Then it is easy to verify that $I$ satisfies the conditions of Lemma 1 and that $I \cap S_{\beta} = \emptyset$. Thus, $S_{\beta}$ cannot be $\varepsilon$-deadly for $G(F, \varepsilon)$, by Lemma 2.

Lemmas 3 and 4 together imply that $G(F, \varepsilon)$ has an $\varepsilon$-deadly set of weight $n$ if and only if $F$ is satisfiable. If $F$ is not satisfiable, every $\varepsilon$-deadly set of $G(F, \varepsilon)$ has weight greater than $n$. Thus, we can decide whether $F$ is satisfiable only by deciding whether $G(F, \varepsilon)$ has an $\varepsilon$-deadly set of weight $n$. An algorithm that computes an $\varepsilon$-deadly set of this weight actually gives a truth assignment $\beta$ which satisfies $F$. As the size of $G(F, \varepsilon)$ is polynomial in the size of $F$, and it takes time polynomial in the size of $F$ to construct $G(F, \varepsilon)$, this proves Theorem 3. The following two corollaries are immediate consequences of Theorem 3.

**Corollary 1** The weighted $\varepsilon$-deadly set problem, $0 < \varepsilon < 1$, is NP-hard for directed acyclic graphs whose nodes have in-degree and out-degree at most 3.
Corollary 2 The unweighted $\varepsilon$-deadly set problem, $0 < \varepsilon < 1$, is NP-hard for directed acyclic graphs whose vertices have in-degree and out-degree at most $2 + \lceil 2/\varepsilon' \rceil$, where $\varepsilon' = \min(\varepsilon, (2 - 2\varepsilon)/\varepsilon)$.

6 Conclusion

Our negative results of Sections 3 and 5 are very strong in the following sense. Our proof in Section 3 implies that for any class of undirected graphs for which the vertex cover problem is NP-hard, the deadly set problem is NP-hard for the corresponding class of directed graphs. As mentioned at the end of Section 5, the construction of that section implies that in general even just deciding whether a given DAG has an $\varepsilon$-deadly set of weight (or size) at most $k$ is NP-hard. An algorithm that computes a minimum deadly set is also able to compute a satisfying truth assignment for a given formula $F$ in 3-CNF.

From a practical point of view, the most important open question is the existence of efficient approximation algorithms. We did not investigate this question.

References


Norbert Zeh is a PhD student at the School of Computer Science, Carleton University, 1125 Colonel By Dr, Ottawa, Ontario, K1S 5B6, Canada. Email: nzeh@scs.carleton.ca

Nicola Santoro is a faculty member at the School of Computer Science, Carleton University, 1125 Colonel By Dr, Ottawa, Ontario, K1S 5B6, Canada. Email: santoro@scs.carleton.ca
Author Index

Baldoni, R., 1
Beier, R., 17
Boldi, Paolo, 37
Bui, Marc, 47
Chen, S.K., 297
Das, Sajal K., 47
Datta, Ajoy K., 47
Dinitz, Y., 63
Dobrev, S., 79
Ducourthial, B., 95
Flammini, Michele, v
Fiocchini, P., 111
Gavoille, Cyril, 127
Georgiou, Chryssis, 141
Héraly, J.M., 1, 181
Harutyunyan, Hovhannes A., 157
Hassin, Y., 167
Hwang, Frank K., 297
Irlande, A., 197
Juurlink, B.H.H., 211
Kolman, P., 211
Konig, J.-C., 197
Královič, R., 111
Laforest, C., 197
Luccio, Fabrizio, 227
Métivier, Y., 237
Melideo, G., 181
Meyer auf der Heide, F., 211
Moran, S., 63
Nakano, K., 253
Nardelli, Enrico, v
Nguyen, Dai Tho, 47
Olariu, S., 253
Pagli, Linda, 227
Peleg, D., 167
Proietti, Guido, v
Rajbaum, S., 63
Raspaud, A., 269
Raynal, M., 1, 181
Rieping, I., 211
Roberts, A., 281
Roncato, A., 111
Ružička, P., 111
Sýkora, O., 269
Santoro, N., 111
Shvartsman, Alex, 141
Sibeyn, J.F., 17
Spirakis, Paul, v
Symvonis, A., 281
Tanguy, L., 237
Tixeuil, S., 95
Vigna, Sebastiano, 37
Vrío, I., 269
Wood, D. R., 281
Zemmari, Akka, 127
Proceedings in Informatics

Carleton Scientific publishes the series *Proceedings in Informatics* as a contribution to the rapid dissemination of high quality results in all aspects and all areas of Informatics, including Computing, Communication, Discrete Mathematics, Algorithms, Complexity, and Networking.

It has as its mandate the publishing of proceedings of conferences, colloquia, workshops and symposia that focus on research themes of scientific importance and relevance. In particular, reflecting the continuous changes in the domain, it actively pursues those meetings devoted to new research topics, emerging areas of scientific investigation, and mutation and integration of existing fields.

The goal of this series is to provide an invaluable working tool for researchers active in (or entering) the specific area covered by each volume, for scientists who want to discover the current directions in Informatics, and for application designers and software engineers who want to keep up to date with the latest results and problems being investigated by the research community.

To achieve this goal and fulfill its mandate, Carleton Scientific has established an international editorial board of scholars working in the area of Informatics.

If you are interested in publishing the proceedings of your meeting in this series, please contact any of the members of the Editorial Board. For any other information contact: editor@carleton-scientific.com

**Series Editorial Board**
Afonso Ferreira (ferreira@ll.inria.fr)
Jose Rolim (Jose.Rolim@cui.unige.ch)
Nicola Santoro (santoro@scs.carleton.ca)

**Managing Editor**
Steve Izma (editor@carleton-scientific.com)