1

of/de

1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Distributed Algorithms for Routing and String Recognition

by

Flaminia L. Luccio

Laurea in Scienze dell'Informazione

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Computer Science**

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

March 8, 1995

© Copyright

1995, Flaminia L. Luccio

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

1 sur file *Votre référence*

Our file *Notre référence*

**THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.**

**L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.**

**THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.**

**L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.**

ISBN 0-612-03005-9

Canada

Name Flaminio LUCCHIO

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

computer science

SUBJECT TERM

0984

U·M·I

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical
Business Administration 0310
General 0272
Accounting 0770
Banking 0454
Management 0338
Marketing 0385
Canadian Studies
Economics 0501
General 0503
Agricultural 0505
Commerce-Business 0508
Finance 0509
History 0510
Labor 0511
Theory 0358
Folklore 0366
Geography 0351
Gerontology
History 0578
General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science
General 0615
International Law and Relations 0616
Public Administration 0617
Recreation 0814
Social Work 0452
Sociology
General 0626
Criminology and Penology 0627
Demography 0938
Ethnic and Racial Studies 0631
Individual and Family Studies 0628
Industrial and Labor Relations 0629
Public and Social Welfare 0630
Social Structure and Development 0700
Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy
Animal Culture and Nutrition 0475
Animal Pathology 0476
Food Science and Technology 0359
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Range Management 0777
Wood Technology 0746
Biology 0306
General 0287
Anatomy 0308
Biostatistics 0309
Botany 0379
Cell 0329
Ecology 0353
Entomology 0369
Genetics 0793
Limnology 0410
Microbiology 0307
Molecular 0317
Neuroscience 0416
Oceanography 0433
Physiology 0821
Radiation 0778
Veterinary Science 0472
Zoology
Biophysics 0786
General 0760
Medical

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Polymology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences
General 0566
Audiology 0300
Chemotherapy 0992
Dentistry 0567
Education 0350
Hospital Management 0769
Human Development 0758
Immunology 0982
Medicine and Surgery 0564
Mental Health 0347
Nursing 0569
Nutrition 0570
Obstetrics and Gynecology 0380
Occupational Health and Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics
Physics 0605
General 0986
Acoustics
Astronomy and Astrophysics 0606
Atmospheric Science 0608
Atomic 0748
Electronics and Electricity 0607
Elementary Particles and High Energy 0798
Fluid and Plasma 0759
Molecular 0609
Nuclear 0610
Optics 0752
Radiation 0756
Solid State 0611
Statistics 0463
Applied Sciences
Applied Mechanics 0346
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal System Science 0790
Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis,
Distributed Algorithms for Routing and String Recognition
submitted by
Flaminia L. Luccio, Laurea in Scienze dell'Informazione
in partial fulfilment of the requirements
for the degree of Master of Computer Science

Director School of Computer Science

Thesis Supervisor

Carleton University
March 27, 1995

Abstract

Communication and cooperation are crucial problems in distributed computing. In this thesis we study aspects of these two problems. First we consider the problem of routing. We present some compact routing techniques, that minimize time and space complexity. We investigate Boolean Routing, and we introduce a new routing function for unidirectional and bidirectional chordal rings with one chord. This function requires $O(n \log n)$ bits and $O(n)$ time. We also study some relations between our technique and Interval Routing Schemes. Next, we consider the problem of recognizing whether a given binary string of length n is equal (up to rotation) to the input of an anonymous, asynchronous ring of n processors. We give new upper and lower bounds on the bit complexity of the problem. For the case of periodic strings, we introduce near optimal bounds which depend on the period of the string. For the case of a randomly chosen string, we show that almost all strings can be recognized by communicating $\Theta(n \log n)$ bits. Kolmogorov complexity theory is here used in the proof of *upper* bounds, rather than its traditional application to the proof of lower bounds.

Acknowledgments

I first would like to thank my supervisor Professor Danny Krizanc whose amazing capability of producing new ideas was a good motivation for my work, and gave me lots of enthusiasm. Thanks also for all his trust and for his moral and financial support.

Thanks to Professor Evangelos Kranakis whom I really enjoyed working with. It is from him that I've learned how to approach a problem and formalize it.

Thanks to Professor Nicola Santoro for his suggestions on the algorithm of Section 3.4.1.

Thanks also to Mike Just for all his helpful hints, and mainly for his patience and moral support.

Thanks to all the other graduate students (in particular to Ravi Palepu and Mike Valenta), and all the people at the School of Computer Science.

Last but not least, thanks to my family, that morally and financially supported me here.

Contents

Abstract	iii
Acknowledgments	iv
List Of Tables	vi
List Of Figures	vii
1 Introduction	1
1.1 Distributed Computing	1
1.2 Routing Algorithms	4
1.3 Computing Boolean Functions	5
1.4 Results of this Thesis	6
2 Routing	7
2.1 Preliminary Definitions	7
2.2 Routing Algorithms	8
2.3 Boolean Routing	10
2.4 Boolean Routing on Chordal Rings	13
2.4.1 Preliminaries	13
2.4.2 Bounds on the Unidirectional Chordal Ring	15
2.4.3 Bounds on the Bidirectional Chordal Ring	22

2.4.4	Routing on Chordal Rings	25
3	Computing Boolean Functions	41
3.1	Introduction	41
3.2	Results of the Chapter	43
3.3	Preliminaries	44
3.4	Upper Bounds	46
3.4.1	Periodic Strings	46
3.4.2	Kolmogorov Random Strings	55
3.5	Lower Bounds	65
4	Conclusion	67
4.1	Future Work	68
A	Index Search	69
	References	75

List of Tables

3.1 Results of Chapter 3.	44
-----------------------------------	----

List of Figures

2.1	A bidirectional chordal ring with $k = 4$ and $n = 19$	14
2.2	A bidirectional chordal ring with $a < \frac{k}{2}$	16
2.3	Case I (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$, and $i_c = 1$	31
2.4	Case II (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$, and $i_c = 1$	31
2.5	Case III (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < k - a \leq k - 1$, and $j_c = 1$	32
2.6	Case IV (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < k - a \leq k - 1$, and $j_c = 1$	32
3.1	The right overlap is of length $0 < l < \frac{t}{2}$, and $n \geq 1 + 2(t - l - 1)$. . .	58
3.2	The right overlap is of length $0 < l < \frac{t}{2}$, and $n < 1 + 2(t - l - 1)$. . .	59
3.3	The left overlap is of length $0 < l < \frac{t}{2}$, and $n > 2t - l + 1$	60
3.4	The right overlap is of length $l \geq \frac{t}{2}$	61
3.5	The left overlap is of length $l \geq \frac{t}{2}$	62
3.6	At least one overlap is of length $l > \frac{t}{2}$	63
3.7	Both overlaps are of length $l \leq \frac{t}{2}$	64

Chapter 1

Introduction

1.1 Distributed Computing

Due to new technological innovations, distributed computing has become increasingly important in the last fifteen years [T92]. Microprocessors have become cheaper and more powerful, and the production of high-speed connection networks has facilitated this need. Local Area Networks (LAN) were the first networks introduced. A LAN is a network of separate computers that can share resources such as printers, disks, application programs, etc. In a LAN the cost of the computers has decreased, communications are fast, and the system is flexible, therefore it is possible to connect new machines. Computation that was once only centralized (the system was using a single CPU) can now be distributed among different processors.

The introduction of Wide Area Networks (WAN) reflects the increasing need for fast communications between distant locations. A WAN is a collection of computers with their own autonomy (no shared memory, devices or applications). The connections are used to exchange information (e.g., e-mail), and to access remote files or databases. The computers are usually located far from each other (e.g., distant buildings or cities), and the connections are point-to-point. Along these lines, distributed systems started to become more powerful.

A distributed system is a set of entities that are autonomous (i.e., have storage and processing ability), and are able to communicate by exchanging messages. These entities can cooperate to solve a single problem. The power of a distributed system comes from the possibility of splitting the work among different entities, i.e., jobs can be parallelized.

We model a distributed system with a graph $G = (V, E)$, where the entities are the nodes (in the set V), and the communication links are the edges (in the set E). The graph may or may not be oriented, and we assume that it is always connected.

We call the system *dynamic* if it structurally changes over time (i.e., nodes/edges are added/deleted), otherwise it is *static*. The system is *symmetric* if every entity executes the same algorithm. The system is *synchronous* (i.e., local and global clocks are synchronized), if the same step of an algorithm is executed at the same time, *asynchronous* otherwise. Every operation is locally executed and the result can be communicated by message passing. In synchronous systems, extra information can also be deduced from the timing of the clock. For example, an entity waits some clock ticks and if no message arrives it can deduce that something is wrong.

We assume that the channels are FIFO (first-in, first-out), and that in the absence of failures the messages will eventually arrive (within a finite amount of time). Each entity communicates with a subset of other entities called *neighbours*, and can distinguish among them (*local orientation*).

The behaviour of an entity depends on external events such as the arrival of messages or clock ticks. The system is defined by a finite set of states. When an event occurs, an entity executes an action that consists of some local processing, the transmission of messages and the changing of its state. The behaviour of the entity is defined by a complete and unambiguous set of rules.

A distributed algorithm is defined by the behaviour of all the distinct entities, and has to be correct (i.e., should solve the problem it has been defined for), efficient (i.e., should use minimal resources), and flexible (i.e., should recover from failures). The execution of an algorithm starts when one or more entities “spontaneously” receive a “wake-up” message (we are not interested in how this happens). The other

entities are then awoken and the real computation starts. The global computation ends when each entity knows the result of the computation.

Communication is a crucial point in a distributed system: why would two entities be connected if they never communicate? At a high level, a global execution of an algorithm is achieved when partial results are exchanged among entities, so communication and synchronization mechanisms have to be defined. This can be done using message passing, and thus implies the need for the study of good communication techniques. Routing is a technique that permits each node to decide the path a message has to follow to reach its destination. If the destination node is a neighbour, the message is forwarded through that link, otherwise an outgoing edge has to be chosen. We are interested in finding good routing algorithms, i.e., algorithms that find shortest paths, and minimize both the space needed to memorize routing information (space complexity) and the time to access it.

Related to communication is the cooperation problem. In a distributed network nodes should cooperate in order to optimize their use of the resources and to speed up computations. A crucial point is how many messages the nodes should exchange in order to reach a mutually agreeable solution to a problem. A further concern is how big should these messages be. In certain regular topologies with particular labeling, some information can be implicitly obtained, i.e., less information needs to be exchanged. However in anonymous networks (i.e., networks in which nodes do not know the labels of others) this cannot be done. In these networks, defining correct and efficient algorithms is not usually trivial, and sometimes impossible. Many algorithms have been presented to solve problems on anonymous networks ([BB89, IR81, KK90, T94]). A limiting factor in these systems is that symmetry cannot be broken, i.e., there cannot be a *leader* (a node that differs from the others). This is a result of the fact that node labels cannot be used to create a distinction among the nodes. Problems like leader election cannot be solved with deterministic algorithms, even if the size of the network is known (see [T94]). In this thesis we investigate a particular problem that can be nicely solved in a particular topology (a ring) with anonymous nodes when the size of the network is known. This is

the string recognition problem on anonymous, asynchronous ring. This problem is defined as follows. Given a ring, each node has as input a single bit and a string. The string is the same for all the nodes. The output of the problem is yes if the string defined by the input bits of the nodes is equal, up to rotation, to the string in the input. We show how the string recognition problem is part of a wider class of problems related to the computation of particular boolean functions in different topologies. Cooperation is crucial to optimally solve the problems. We will focus on the amount of information exchanged to achieve the solution (i.e., how many bits will globally be exchanged in the network).

1.2 Routing Algorithms

Solving a routing problem consists of defining a way in which entities that are physically separated can communicate.

As was previously stated, in a WAN the connections are point-to-point. We assume that the graph representing the network is connected. If two entities that want to communicate are directly connected, the routing is straightforward, since the message is sent on the edge that connects the two. If the two entities are not directly connected a path between them has to be found. Routing techniques differ depending on what kind of information has to be stored in the nodes and in the messages. We assume that labels are associated to nodes and edges: a destination address coincides with a node label. A good routing algorithm follows these rules:

- it is correct (gives an admissible solution);
- is fair (serves each node equally);
- optimizes space (to memorize node/edge labels and routing functions) and time (to access the information);
- searches for shortest paths (in weighted networks), or minimum hops paths (in unweighted networks);

- minimizes traffic congestion (giving alternative paths);
- is robust (it allows topological changes);
- avoids deadlocks.

All these points are not necessarily considered at the same time. Recently, particular attention has been given to the study of some techniques (called *Compact Routing techniques*) whose principal aim is to optimize the trade-off between the length of the paths and the space required for routing. Later we will discuss some of these techniques that search for shortest paths and try to optimize the space. These are Interval Routing Schemes ([BLT91, FG94, FJ88, FGS94, KKR93, SK85, LT87]), Prefix Routing Schemes ([BLT90]), Compact Routing ([CL92]), and Boolean Routing ([FGS94]). We study Boolean Routing in detail, analyze the time/space required for routing, and we will also show how this method can apply to a particular topology of interest: a chordal ring. We will also extend and generalize some results in ([FGS94]), where the authors present Interval Routing Schemes on some particular chordal rings.

1.3 Computing Boolean Functions

As we have mentioned, it is interesting to study the power of cooperation in a communication network, and also to see the amount of information that has to be exchanged to achieve it. What we are interested in, is the problem of computing boolean functions on anonymous, asynchronous rings, where the size of the ring is known. By computing a boolean function we mean that the input at each processor is a value in $\{0, 1\}$, and the output is a boolean function of this initial state. Since achieving the solution requires that every processor has to collect some kind of information, the complexity is defined in terms of the number of messages that have to be exchanged among the processors. In particular we are interested in the total number of bits exchanged. In [ASW88] it is shown that all functions are computable on an anonymous, asynchronous ring of n processors using $O(n^2)$ bits, and a function

that requires as many bits is given. Furthermore it is shown that almost all boolean functions on n variables, require $\Omega(n^2)$ bits to compute. On the other hand, in [MW86] an $\Omega(n \log n)$ lower bound for computing any non-constant function, and a function requiring $O(n \log n)$ bits are given. We will present an algorithm that solves the string recognition problem and therefore computes the output of the boolean function associated to it. Using some properties of Kolmogorov complexity theory, we will prove that the algorithm requires $\Theta(n \log n)$ bits for almost every string.

1.4 Results of this Thesis

The remainder of this thesis is organized as follows.

In Chapter 2 we describe the routing problem in detail, and we present some compact routing techniques, giving particular attention to Boolean Routing. We also introduce a new Boolean Routing function for unidirectional and bidirectional chordal rings with one chord, and we show that the algorithm presented has a $O(n)$ time and $O(n \log n)$ space complexity in the worst case.

In Chapter 3 we describe some general results on algorithms for computing boolean functions on anonymous, asynchronous rings, and we introduce a near-optimal algorithm to compute a new function that solves the string recognition problem. The results of this chapter have been presented in [KKL94].

In Chapter 4 we make some final observations and we present some open problems.

Chapter 2

Routing

The routing of a message in a network of processors consists of several steps. Given that a processor is a node and a connection is an edge of a graph, when a node receives a message it executes the following steps: 1) it takes the destination address as an input; 2) it stops if it is the destination, or it outputs the next edge the message has to traverse on its way to its final destination. Information to perform this operation must be stored in the node and derived in real time.

In this chapter we are interested in studying techniques that define minimum hop paths in an unweighted graph (in some cases this techniques can be extended to weighted graphs), and that minimize the time and space required for the routing.

The following sections are organized as follows. In Section 2.1 we give some preliminary definitions. In Section 2.2 we present some routing techniques, and in Section 2.3 we describe Boolean Routing in detail. Finally, in Section 2.4 we introduce a new boolean function for unidirectional and bidirectional chordal rings with one chord. We also introduce some nice relations between our boolean technique and a different routing technique: Interval Routing Schemes.

2.1 Preliminary Definitions

We consider a graph $G = (V, E)$ where V is the set of the nodes, $|V| = n$, and E is the set of edges. For each node i , $i = 1, \dots, n$, d_i is its out-degree, i.e., the number

of outgoing edges, and $d = \max_{i=1, \dots, n} \{d_i\}$. We assume that the graph is connected so that every pair of nodes is able to communicate. δ is the diameter of the graph, i.e., the maximal distance between two different nodes.

2.2 Routing Algorithms

Different routing techniques can be defined by assuming that different kinds of information is stored in the messages or in the nodes. The cost of routing is defined in terms of the space complexity, i.e., the number of bits used, and in terms of the time required to find the outgoing edge. An example of a simple but expensive (in terms of space) routing technique is the following: a complete routing table is memorized at each node of the graph, and it contains for each destination node an edge that is on a shortest path to it. These tables are defined before the computation starts, and have to be dynamically changed if there is a failure in the system. When a node receives a message it checks whether it is the destination node, and if not, it checks in the table on which edge the message should be forwarded. Each message contains both the information to be exchanged and the destination address.

The time to access the table is constant, but the space required is large since it is given by the size of the message, plus the size of the table which is $O(n \log d)$ bits. There are $n - 1$ possible destinations, and for each one we associate an outgoing edge that is on a shortest path to it (we need $\log d$ bits to identify the edge). The total space complexity over all nodes is then $O(n^2 \log d)$ bits. If we extend this to a table that contains complete information on all the shortest paths, $O(nd)$ bits are required in each node. There are again $n - 1$ destinations, and for each one d bits are required. Among these d bits there is a 1 if the corresponding outgoing edge is on a shortest path to the destination, otherwise the bit is 0. In a complete bipartite graph for instance $O(n^3)$ bits are required to define all the shortest paths if this method is used.

Compact Routing techniques optimize the time and space complexity by using some implicit information in the network (for example, the labeling of the nodes).

The first example of an implicit routing technique for arbitrary graphs was presented in 1985 by Santoro and Khatib [SK85]. Optimal (in the sense that shortest paths are defined) implicit solutions had previously been found for particular regular topologies such as meshes, hypercubes ([W81]), etc. In [SK85] the authors define an implicit routing algorithm that is optimal for acyclic graphs and has a worst-case complexity that is optimal within a factor of two compared to other graphs. This method will later be referred to as the *Tree-labeling Scheme*.

The Tree-labeling Scheme was extended in 1987 by van Leween and Tan [LT87] to non-tree networks, and was called an *Interval Labeling Scheme* (ILS). An ILS consists of giving a different label from $\{1, \dots, n\}$ to each node of the graph, and associating one or more intervals to each edge of the graph. If each edge contains at most one interval we have a 1-ILS. If the maximum number of intervals on an edge is k , we have a k -ILS. Furthermore if the interval has "wrap-around", then the method is called an Interval Routing Scheme (IRS), otherwise is called a Linear Interval Routing Scheme (LIRS).

In a 1-ILS the intervals are defined in such a way that node j is in an interval associated to an edge outgoing from node i , if and only if that edge is on a shortest path from i to j . Furthermore, the union of the outgoing intervals from i covers all nodes in the graph, and the intersection is disjoint. This means that a unique shortest path is defined from each source to each destination node.

A 1-ILS can easily be defined for simple topologies such as trees, rings, hypercubes, meshes, some chordal rings, every arbitrary acyclic network, etc. ([FGS94, FG94, SK85, LT87]). A 1-LIRS is defined for a more restricted set of topologies [BLT91, FG94].

A limit of this method is that in presence of node/edge failures the routing information has to be redefined. Traffic congestion is also hard to avoid. A solution to the last problem could be the definition of alternative paths to be used if the system is overcrowded.

In terms of memory this technique is better than the complete routing tables since a generic k -ILS or k -LIRS require $O(nkd \log n)$ bits. Since k is often a constant, this

techniques has better space complexity.

Prefix Routing Schemes (PRS) were first presented in 1990 by Bakker, van Leeuwen and Tan [BLT90]. The *Prefix Labeling Scheme* (PLS) consists of giving a different label (a string from Σ^* for an alphabet Σ) to each node of the graph, and on associating a label (a string from Σ^* for an alphabet Σ) to each edge of the graph. Each edge label is a prefix of the label of the node to which the edge is connected.

The routing is very simple. If a node is not the destination, it forwards the message on the edge whose label is the longest prefix of the address (if the routing has been correctly defined there must be at least one edge with this property). All the information regarding the edges is stored in a compact routing table.

PRS work well on dynamically growing networks. The addition of a new node requires only a new node label and the defining of the labels in the new edge (that is the routing table memorized in the two nodes), i.e., the adaptation cost is constant. Further, a PRS can be defined for every connected graph (the paths are of length at most two times the diameter). In terms of space, $O(nd\delta \log \sigma)$ bits are necessary, if σ is the number of symbols of Σ .

Compact Routing (see [CL92]) requires more information in the message header. For example, the message contains the whole path between the source and the destination (a concatenation of all the node labels). During the routing, part of the header is removed. If the diameter of the graph has length δ (in general it will be $O(\log n)$), then the header contains $\log d\delta$ bits. The path of a message has to be defined by a translation table. This table can be larger than a standard routing table, but is small in networks in which only a small subset of nodes communicate, or in regular networks where the path can somehow be deduced from the source and destination nodes.

2.3 Boolean Routing

Boolean Routing techniques were first introduced in 1993 by Flammini, Gambosi, Salomone [FGS94]. We are interested in explaining it in more detail since some of

the results in this chapter are strictly related to it.

The first important property of Boolean Routing Schemes (BRS) is that they define all the shortest paths among two different nodes. For this reason failures can be easily recovered from (there are more chances to isolate them since alternative paths can be found), and traffic congestion can be decreased. Note that it is *NP*-hard to derive an IRS with a minimum overall number of intervals when all the shortest paths are searched [FGS94]. Moreover, an IRS can always be simulated by a BRS using the same amount of information ([FGS94]). In the following sections we will introduce topologies for which an IRS has not been defined, but a simple BRS exists.

The BRS does not require any routing table, since it is based on the definition of particular boolean predicates (one for each directed edge). The destination address of a message is given at run-time as input to the function. The output of the function is true if and only if the corresponding edge is on a shortest path from the source to the destination. The functions presented in [FGS94] are very simple, such as the equality function, the threshold functions or a combination of both.

The space complexity of the BRS is defined in terms of the space needed to represent these functions. The functions defined in [FGS94] are optimal for the networks presented below, since they match the lower bound of $\Omega(n \log n + \sum_{i=1}^n d_i \log d_i)$. $\Omega(n \log n)$ is the space required by each node to represent the equality function (every node has to check whether it is the destination). The remaining $\Omega(d_i \log d_i)$ are required in node i to assign distinct labels to the edges.

In [FGS94] no assumption is made on the model that is used to measure the time complexity. We will assume that the computation is done on a model where the operations such as equality and threshold functions, addition, subtraction, etc. can all be done in constant time. The space complexity (as we have previously said), is measured in terms of the bits required. These assumptions will be useful when we will compare the results in this section with those in the next section.

The BRS can be easily defined in very regular topologies.

- **Paths:** an increasing label is assigned to each node from left to right. When

a message arrives, the address is first compared to the receiving node label (equality function). If the labels are different the message is routed to the left if the destination address is smaller, otherwise to the right (threshold function). Since there are only two functions and they are evaluated over the labels, the overall complexity is $O(n \log n)$ bits.

- **Rings:** labels are assigned using a counter-clockwise direction and by concatenating a 0 plus a number from 0 to $\lceil \log n \rceil$ in front of the first half and a 1 plus a number from 0 to $\lceil \log n \rceil$ in front of the others. The routing function is defined in such a way that the message travels left or right depending on the position of the destination. The functions are a composition of simple functions and the complexity is $O(n \log n)$ bits. If the number of nodes in the ring is even, one node is reachable from both sides. Moreover, the function can be extended to the case of weighted links.
- **Trees:** nodes are labeled with a preorder visit. A boolean function given as a composition of threshold functions can be defined. The cost is $O(n \log n)$ bits.
- **d -dimensional Hypercubes:** If $n = 2^d$, each node is connected to $d = \log n$ other nodes. Labels are assigned in such a way that two adjacent nodes differ by at most one bit. Routing is done by comparing one bit at a time starting from the right. The space complexity is $O(n \log n \log \log n)$ bits since $O(\log \log n)$ bits are necessary to identify a different predicate on each link, plus each node stores $\log n$ predicates.
- Simple routing functions can be defined for $n \times m$ grids (with $O(n m \log n m)$ bits); d -dimensional grids where $n_1, \dots, n_d$ are the numbers of nodes in the different directions (with $O(n \log n + n d \log d)$ bits, $n = n_1 \times n_2 \cdots \times n_d$); complete and complete bipartite graphs (with $O(n^2 \log n)$ bits).

It is easy to prove that for each of these networks the upper bound reaches the lower bound (just substitute the corresponding values for d , in all the different cases).

2.4 Boolean Routing on Chordal Rings

Our main contribution in this chapter is the extension of the Boolean Routing technique to the case of unidirectional and bidirectional chordal rings with one chord.

We consider the case of unweighted graphs and we assume that there are no failures in the system. We are interested in finding at least one shortest path from a source node u to a destination node v . (A shortest path in this case is a path from u to v that minimizes the total number of edges to be traversed.) All the results in this chapter can be extended to find all the shortest paths.

The next sections are organized as follows: in Section 2.4.1 we give the introductory definitions for chordal rings and a new algorithm called *Greedy_i*. In Section 2.4.2 and 2.4.3 we define some properties of the algorithm in the case of unidirectional and bidirectional chordal rings with n nodes. Finally, in Section 2.4.4 we show how the previous results can apply to define a boolean routing function with worst case time complexity of $O(n)$ steps, and space complexity of $O(n \log n)$ bits. We also show some relations between *Greedy_i* and ILS.

2.4.1 Preliminaries

Let us formally define what we mean by a bidirectional chordal ring.

Definition 2.1 [FGS94]: Given a ring $G = (V, E)$, where $V = \{v_0, \dots, v_{n-1}\}$, and $E = \{(v_i, v_{(i+1) \bmod n}) \mid i = 0, \dots, n-1\}$, and given a set $I \subseteq \{2, \dots, n-2\}$, a chordal ring $C = (V, E', I)$ is an undirected graph with $E' = E \cup \hat{E}$, where $\hat{E} = \{(v_i, v_{(i+j) \bmod n}) \mid i = 0, \dots, n-1, j \in I\}$. The edges in $\hat{E}$ are called chords.

We can extend this definition to unidirectional graphs, by assuming that the edges are pointing in a clockwise direction. We restrict our study to the set of *chordal rings with one chord* $C = (V, E', \{k\})$ (see Figure 2.1). Moreover, we consider the case $k \leq \frac{n}{2}$ since if $k > \frac{n}{2}$ we just need to substitute k with $n - k$ in all the equations and we obtain the same results. Given a unidirectional or bidirectional chordal ring, we

Figure 2.1: A bidirectional chordal ring with $k = 4$ and $n = 19$

are interested in finding all shortest paths between a source node u and a destination node v . W.l.o.g., we can look for a single shortest path (it is trivial to extend the function to another one that computes all shortest paths). We also assume that the source node is 0 and the other nodes have cyclic integer labels.

Definition 2.2 : Given a bidirectional chordal ring $C = (V, E', \{k\})$, and given a source node 0, we denote the outgoing edges of 0 as: $e_1 = (0, 1)$, $e_2 = (0, k)$, $e_3 = (n - 1, 0)$, $e_4 = (n - k, 0)$.

NOTATION 2.1 : Given a chordal ring $C = (V, E', I)$ as in Definition 2.1, we say that we “jump i times around the ring” if, starting from a source node u , we cross the node u i times, while moving along the ring in the same direction.

We define an algorithm $Greedy_i$ for this problem as follows: in the case of a unidirectional chordal ring, $Greedy_0$ consists of moving with jumps of length k to the first node p whose distance to the destination node v is $< k$, moving in a clockwise direction. Then we move along the ring with steps of length 1. $Greedy_i$ consists of moving to the same node p and then moving (with respect to the node p) with jumps of length k , additional i times around the ring in a clockwise direction. At this point we move along the ring to the destination node v . (Note that $Greedy_0$ coincides with the standard greedy algorithm.)

W.l.o.g., we consider the node 0, and we calculate the first i points included in the interval $[0, \dots, k]$, that we can reach by jumping with steps of length k one or more times around the ring. The shortest path will be given by the shortest path that passes through 0 or through one of these i points, and that then moves along the ring in a clockwise direction only (it is easy to calculate these distances). The integer i is the biggest value for which at least one node is optimally reached with i jumps around the ring.

In the bidirectional case we can move both in the clockwise or in the counter-clockwise direction. This means that the set of points we have to consider is given by the closest node (one for each direction) and by a set of $2i$ points. The distances have to be calculated by considering moves in both directions.

To facilitate the reading of the paper, we define the following:

NOTATION 2.2 : *The points that we reach going 1 time, 2 times, ..., s times around in a clockwise direction and that fall in $[0, k)$, will be denoted with the quantity $a, 2a, \dots, sa$. Likewise $b, 2b, \dots, sb$ are the points we reach going in a counter-clockwise direction. This time though b is reached one step sooner (see Figure 2.2). With another step we reach $n-a$, etc. The set of points associated with unidirectional Greedy_i is $A = \{a, 2a, \dots, ia\}$, whereas those associated with bidirectional Greedy_i is $B = \{a, 2a, \dots, ia, b, 2b, \dots, ib\}$. Moreover, if $k < \frac{n}{2}$ then $a = k(\lfloor \frac{n}{k} \rfloor + 1) - n$ and $b = n - k\lfloor \frac{n}{k} \rfloor$; if $k = \frac{n}{2}$ then $a = 0$ and $b = \frac{n}{2}$.*

We define an optimal index for Greedy_i as:

Definition 2.3 : *The index i of Greedy_i is optimal iff there exists at least one node in $[0, k]$ whose shortest path from 0 requires at least i jumps of length k around the ring, and no shortest path to any other node in $[0, k]$ requires more than i jumps around the ring.*

2.4.2 Bounds on the Unidirectional Chordal Ring

In this section we present some results for the case of the unidirectional chordal ring. We give a straightforward upper bound on the optimal index i of Greedy_i and we

Figure 2.2: A bidirectional chordal ring with $a < \frac{k}{2}$.

show a class of chordal rings for which this bound is tight. Moreover, we present an algorithm that calculates the optimal i , and that has an $O(n)$ time complexity. The space complexity is $O(\log n)$ bits.

There are some results related to the upper bound of $Greedy_i$ that are easy to derive.

Theorem 2.1 : *Given a unidirectional chordal ring, an upper bound on the optimal index of $Greedy_i$ is given by $m = \left\lfloor \frac{k(k-1)}{n} \right\rfloor$.*

PROOF Since we can move at most $k-1$ steps with jumps of length k (with at most k steps of length 1 we can reach all the points in the interval $[0, k]$ starting from 0), an upper bound on i is given by $\left\lfloor \frac{k(k-1)}{n} \right\rfloor$, that is, the number of times we cross node 0. ■

The upper bound m is attained if the distance we have moved with $Greedy_m$ is smaller then the label of the node we have reached and is smaller then the distance of each path passing through $Greedy_j$ with $j < m$.

There are chordal rings for which the upper bound is tight, others for which is not.

Theorem 2.2 : *There exists an infinite class of unidirectional chordal rings for which the upper bound m is attained.*

PROOF To prove this we will build up one of the possible class of infinite examples of unidirectional chordal rings. We consider the case of *Greedy*₀, i.e., the case for which the upper bound is $m = \left\lfloor \frac{k(k-1)}{n} \right\rfloor = 0$. (It is even possible to do this with a generic *Greedy*_i by adding other new conditions.) The upper bound is reached if jumping one time around the ring, with steps of length k we reach a point $0 \leq a \leq \left\lfloor \frac{n}{k} \right\rfloor + 1$.

In other words, given two integers n and $1 < k < n$ the upper bound 0 is reached by the infinite class of graphs for which:

$$k(k-1) < n$$

and

$$0 \leq k \left(\left\lfloor \frac{n}{k} \right\rfloor + 1 \right) - n \leq \left\lfloor \frac{n}{k} \right\rfloor + 1$$

hold. ■

There are chordal rings for which the upper bound is not attained, i.e., we have the following:

Theorem 2.3 : Given $k = sa + r$, $a \leq \frac{k}{2}$ ($k = s(k-a) + r$, $a > \frac{k}{2}$), for k, s, r integers, $0 \leq r < a$ (or $0 \leq r < k-a$), $a = k(\left\lfloor \frac{n}{k} \right\rfloor + 1) - n$ ($a = 0$ for $k = \frac{n}{2}$), the upper bound is not attained if $m > s$.

PROOF If $a = 0$ and $k = \frac{n}{2}$ the upper bound $m = \left\lfloor \frac{\frac{n}{2}-1}{2} \right\rfloor$ is not optimal independent from the value of s . We assume than that $k < \frac{n}{2}$. We first study the case $s < m \leq 2s$ if $r \leq \frac{a}{2}$, or $s < m \leq 2s+1$ if $r > \frac{a}{2}$. We now show that this upper bound m cannot be optimal. There are four cases, namely:

1. $k = (s+1)a$, that is the optimal i is given by:

if $a > \left\lfloor \frac{n}{k} \right\rfloor + 1$ then $i = s$ else $i = 0$. No $m > s$ can be optimal since the new points we would find had previously been found.

2. $a < \frac{k}{2}$ (that is $s > 1$), $0 < r \leq \frac{a}{2}$. First observe that if we optimally reach a node with $s+1$ jumps, then we also optimally reach a node with $s+2$, up to

$2s$ jumps (if $s+1 < 2s$). We can limit our study then to the case for which $2s$ is the optimal index. This case occurs when all the following relations hold:

$$(s-1) \left\lfloor \frac{n}{k} \right\rfloor + s - 1 + a - r > 2s \left\lfloor \frac{n}{k} \right\rfloor + 2s - 1,$$

$$2s \left\lfloor \frac{n}{k} \right\rfloor + 2s - 1 < k,$$

$$(2s+1) \left\lfloor \frac{n}{k} \right\rfloor + 2s \geq k.$$

The first relation states that it is more convenient to jump $2s$ then s times around the ring, the other two that m is the upper bound. By easy calculations this implies that $a - r = k \left\lfloor \frac{n}{k} \right\rfloor + k - n - r > (s+1) \left\lfloor \frac{n}{k} \right\rfloor + s \geq \frac{k}{2}$ which cannot be since $a < \frac{k}{2}$.

3. $a < \frac{k}{2}$, $r > \frac{a}{2}$. Prove similar to the one above.

4. $a > \frac{k}{2}$, i.e., $k = s(k-a) + r$. In this case again we can limit the attention to the value $m = s+1$. The relations are then:

$$\left\lfloor \frac{n}{k} \right\rfloor + 1 + r > (s+1) \left\lfloor \frac{n}{k} \right\rfloor + 2,$$

$$(s+1) \left\lfloor \frac{n}{k} \right\rfloor + 2 < k,$$

$$(s+2) \left\lfloor \frac{n}{k} \right\rfloor + 2 \geq k.$$

With easy calculations we see that i cannot be optimal.

If $m > 2s$ for $r \leq \frac{a}{2}$ or if $m > 2s+1$ for $r > \frac{a}{2}$, all the considerations made before hold. In case 1) we will reach again points we have found before, whereas in case 2), 3), 4) we will fall in intervals previously defined and the distances between old and new points decreases. That is if m was not optimal before, it cannot be optimal now. ■

There are certain graphs for which we can easily calculate the optimal value i , so that we don't need the upper bound m .

Theorem 2.4 : *Given a unidirectional chordal ring of n nodes, and given $k \leq \frac{n}{2}$, the following properties hold:*

1. *Greedy₀ is optimal iff $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$.*
2. *Greedy₁ is optimal iff $(a \geq \lfloor \frac{k}{2} \rfloor)$ and $(a > \lfloor \frac{n}{k} \rfloor + 1)$ and $(2a \leq 2\lfloor \frac{n}{k} \rfloor + 1)$.*
3. *If $k \bmod a = 0$ and $a \leq \frac{k}{2}$, then Greedy_($\frac{k}{a}-1$) is optimal.*
4. *If $(k - a = b)$ and $(n \bmod b = 0 \text{ or } \lfloor \frac{n}{k} \rfloor \bmod b = 0)$, and $(b < \lfloor \frac{k}{2} \rfloor)$, then Greedy_($\lfloor \frac{k}{2} \rfloor - \frac{b}{r}$) is optimal.*

PROOF Case 1): If $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ it is faster to move along the ring then to go one time around with jumps of length k . Moreover, if we keep on going around we still have that all the shortest path from 0 to $[0, k]$ are still around the ring (trivial). On the other hand if $a > \lfloor \frac{n}{k} \rfloor + 1$ there exist at least one node whose shortest path is obtained by going at least one time around with jumps of length k , i.e., the optimal i of Greedy _{i} is ≥ 1 .

Case 2): If $a = \lfloor \frac{k}{2} \rfloor$ and we keep on jumping around the ring with jumps of length k we reach again only the nodes 0, $\lfloor \frac{k}{2} \rfloor$, k . Since $a > \lfloor \frac{n}{k} \rfloor + 1$ then $i = 1$.

If $0 \leq a < \lfloor \frac{k}{2} \rfloor$ and $a > \lfloor \frac{n}{k} \rfloor + 1$ then $i > 1$ else if $a \leq \lfloor \frac{n}{k} \rfloor + 1$ then $i = 0$.

The last case is $a > \lfloor \frac{k}{2} \rfloor$. If $a > \lfloor \frac{n}{k} \rfloor + 1$ then $i \geq 1$. Since $2a \leq 2\lfloor \frac{n}{k} \rfloor + 1$ all the nodes up to $2a$ have a shortest path starting from 0 and moving along the ring. For ia , $i > \lfloor \frac{k}{k-a} \rfloor$ the best path is from a , and so on, i.e., $i = 1$ is optimal.

If $i = 1$ this means that we want to reach only one point in $[0, k]$ with jumps of length k . It is easy to see that the set defined above is complete in this respect.

Case 3): The first $\frac{k}{a} - 1$ jumps reach optimally the $\frac{k}{a} - 1$ points. In the next jumps we fall in points already found before.

Case 4): The idea is the same as before, but this time we might be able to reach faster some of the points starting from 0 since we are moving backwards ($b < \lfloor \frac{k}{2} \rfloor$). The fact that b divides n or $\lfloor \frac{n}{k} \rfloor$ implies that we jump again over points we have reached before, i.e., at most $\lfloor \frac{k}{2} \rfloor - \frac{b}{r}$ jumps could reach optimally some points. ■

We can check these relations with a constant number of operations.

If we do not fall in any of these cases we have to use a general technique (see the Index Search algorithm in the Appendix) that gives us as output the optimal value of the index i .

There are two cases: 1) $a < \frac{k}{2}$, 2) $a > \frac{k}{2}$. (The case $a = \frac{k}{2}$ is handled in Theorem 2.4 case 3.) In both cases the algorithm ends after the number of moves is $> k - 1$ (see proof of Theorem 2.1).

In the first case we calculate the value of i by considering only the first subinterval $[0, a]$, since the behavior in the other parts is similar. We maintain a list of indices that are considered to be the number of jumps required to reach different nodes, with a shortest path during the previous steps. Whenever we get a new value j we have to check whether the shortest path to the node we reach with j jumps requires all these j jumps or less. The following proposition holds (in both cases 1 and 2):

PROPOSITION 2.1 : *A unidirectional chordal ring is given. Assume that jumping i times around the ring we reach a node x . Assume moreover that this gives us a path from 0 to x that is shorter than any other path we get jumping $0 \leq z < i$, $i \geq 1$ (no other path if $i = 0$) times around and then moving along the ring. It cannot happen that if we jump $j > i$ times around the ring we reach the node x with a smaller number of steps.*

PROOF

- Assume $0 \leq a \leq \frac{k}{2}$. Moreover, we have optimally reached node ia with *Greedy_i* in d_{ia} jumps (for optimal in respect to node ia we intend that the path we have found jumping i times around the ring is shorter than any other path with $0 \leq z < i$ jumps). We now prove that given a $j > i$, and given the node c such that $0 \leq c < ia$ and such that is optimally reached with *Greedy_j* in d_c jumps, it is not possible to reach node ia through c with less steps than with *Greedy_i*.

We have $a > \lfloor \frac{n}{k} \rfloor + 1$ and $a \leq \frac{k}{2}$. Moreover, we have $d_c \geq d_{ia} + \lfloor \frac{n}{k} \rfloor$. d_{ia} is not optimal if $d_{ia} > d_c + ia - c$, but substituting the previous relations we get $d_{ia} > d_{ia} + \lfloor \frac{n}{k} \rfloor + ia - c$ which is false.

- If $a > \frac{k}{2}$ we use the same idea as above. ■

Thus no update of the old values is required. In order to speed up the algorithm, before any comparison with the values of the old j' we check whether the number of steps associated with j is smaller then the distance along the ring from 0 (we know that 0 is always an optimal value). The data structure that we use is an array of dimension $3 * (a + 1)$. We insert in $O(1)$ time every value we suspect to be optimal. (One field is used for the distance, one for the index i associated and the last one for the index j corresponding to the suspected optimal index in the jumps we do at the same level, i.e., before going around with a jump of $\lfloor \frac{n}{k} \rfloor$.) The initialization of the array costs $O(a)$ (steps of time) at the beginning. We stop when we have moved a total distance of $\leq k - 1$. Then we scan the array starting from 0 and we compare each value $\neq 0$ with the previous and eventually we update it with the distance in the shortest path, the new index i and j . Moreover, we keep track of the biggest optimal j that we have found so far. At the end this will be the final i .

The second case is slightly different since the jumps are towards the node 0. This means that an index that is optimal in a round in the last interval, that is $[k - a, k]$, might not be optimal in the other intervals. To avoid this problem we calculate the optimal i in this interval as for the other case. If x is the value of the node corresponding to i , the optimal i is at the end given by the value $i + \lfloor \frac{x - G[0, x] - \lfloor \frac{n}{k} \rfloor - 1}{a} \rfloor$. ($G[0, x]$ is the distance of the shortest path of x from 0.) This means that we consider an i at the same round only if it is not convenient to walk along the ring. The data structure in this case is an array $2 * (k - a + 1)$, since the last field is not required.

Theorem 2.5 : *The time complexity of Index Search is $O(a) + O\left(\frac{k}{\lfloor \frac{k}{a} \rfloor \lfloor \frac{n}{k} \rfloor}\right)$ steps if $a < \frac{k}{2}$, or $O(k - a) + O\left(\frac{k}{\lfloor \frac{k - a}{k} \rfloor \lfloor \frac{n}{k} \rfloor}\right)$ steps for $a > \frac{k}{2}$. Thus in the worst case the time complexity is $O(n)$ steps.*

PROOF The time complexity of the algorithm is given by the cost of initializing and updating the array, plus the cost of the cycle (in which we execute constant costs

operations). That is for $a < \frac{k}{2}$ we have $O(a) + O\left(\frac{k}{\lfloor \frac{k}{2} \rfloor \lfloor \frac{n}{k} \rfloor}\right)$ steps, (for $a > \frac{k}{2}$ we have $O(k - a) + O\left(\frac{k}{\lfloor \frac{k}{2} \rfloor \lfloor \frac{n}{k} \rfloor}\right)$ steps respectively).

In the worst case however, the time complexity is $O(n)$ steps. For example for $k = \lfloor \frac{n}{2} \rfloor - 1$ and $a = \lfloor \frac{k}{2} \rfloor - 1$, (for $k = \lfloor \frac{n}{2} \rfloor - 1$ and $k - a = \lfloor \frac{k}{2} \rfloor - 1$ respectively). ■

2.4.3 Bounds on the Bidirectional Chordal Ring

In this section we present some results for the case of the bidirectional chordal rings. By definition of the *Greedy_i* algorithm on unidirectional and bidirectional ring, it is easy to see that the upper bounds for the unidirectional case hold on the bidirectional case. We can tighten the bound with the following:

Theorem 2.6 : *Given a bidirectional chordal ring, an upper bound on the optimal index of Greedy_i is given by $m = \left\lfloor \frac{k \lceil \frac{k}{2} \rceil}{n} \right\rfloor$.*

PROOF As the unidirectional case, but considering the fact that we can reach node k in one step and then move in the counter-clockwise direction along the ring. ■

We have the following:

Theorem 2.7 : *There exist an infinite class of bidirectional rings for which the upper bound m is attained.*

PROOF The proof is similar to the one for the unidirectional case (see proof of Theorem 2.2). The only difference is that now we can jump backwards. This means that given two integers n and $1 < k < n$ the upper bound 0 is reached by the infinite class of graphs for which:

$$k \left\lceil \frac{k}{2} \right\rceil < n$$

and

$$k \left(\left\lfloor \frac{n}{k} \right\rfloor + 1 \right) - n \leq \left\lfloor \frac{n}{k} \right\rfloor$$

hold. ■

A precise study can be made on the class of graphs for which the optimal index is 0 or 1. We first observe that, as in the case of unidirectional ring, if $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ then $i = 0$.

Corollary 2.1 : *Greedy₀ is optimal iff $(n \bmod k = 0)$ or $(n \bmod (n - k) = 0)$ or $(0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1 \text{ and } a \leq \frac{k}{2})$ or $(0 \leq b \leq \lfloor \frac{n}{k} \rfloor \text{ and } a > \frac{k}{2})$.*

PROOF If $n \bmod k = 0$ ($n \bmod (n - k) = 0$ respectively) after $\frac{n}{k}$ ($\frac{n}{n-k}$) steps of length k we end up in node 0 again, that is it was not worth to jump around. The other cases are straightforward. Moreover, the set of cases is complete since in each other case it is easy to prove that it is better to jump at least one time around the ring. ■

A complete study of *Greedy₁* can be easily done.

Theorem 2.8 : *Greedy₁ is optimal iff*

1. $a > \lfloor \frac{n}{k} \rfloor + 1$ and $a < b$ and $((2a \leq k \leq 2a + 2\lfloor \frac{n}{k} \rfloor + 1)$ or $(3a - \lfloor \frac{n}{k} \rfloor \leq k \leq 3a + \lfloor \frac{n}{k} \rfloor))$;
2. $a > \lfloor \frac{n}{k} \rfloor$ and $a = b$;
3. $b > \lfloor \frac{n}{k} \rfloor$ and $a > b$ and $((2b \leq k \leq 2b + 2\lfloor \frac{n}{k} \rfloor)$ or $(3b - \lfloor \frac{n}{k} \rfloor \leq k \leq 3b + \lfloor \frac{n}{k} \rfloor))$.

PROOF In case 1) and 3) we have to show that point a or b or both have to be reached through the chords (this is implied by $a > \lfloor \frac{n}{k} \rfloor + 1$ and $a < b$, in case 1), or by $b > \lfloor \frac{n}{k} \rfloor$ and $a > b$ in case 3)), but not point $2a, 2b, 3a, 3b, \dots$

case 1): We first study the intervals in which $2a$ and $2b$ have to fall, in order to be sure that we will not reach them walking only along the chords, but from point 0, or a or b or k , that is at least one relation from each pair should hold:

1. $b - \lfloor \frac{n}{k} \rfloor - 2 \leq 2a \leq b + \lfloor \frac{n}{k} \rfloor + 2, k - 2\lfloor \frac{n}{k} \rfloor - 1 \leq 2a \leq k$;
2. $a - \lfloor \frac{n}{k} \rfloor \leq k - 2a \leq a + \lfloor \frac{n}{k} \rfloor, 0 \leq k - 2a \leq 2\lfloor \frac{n}{k} \rfloor + 1$.

We combine the two relations. Since the second one is the same in point 1 and 2, i.e., it is sufficient that either

$$2a \leq k \leq 2a + 2\left\lfloor \frac{n}{k} \right\rfloor + 1$$

or

$$3a - \left\lfloor \frac{n}{k} \right\rfloor \leq k \leq 3a + \left\lfloor \frac{n}{k} \right\rfloor$$

hold. What we still have to prove is that, although we are analyzing the intervals that cover $2a$ and $2b$, this is sufficient to cover all the ia and ib , for $i \geq 3$. We will not give further details, but it is easy to see that starting from $0, k, a, b$, we can cover all the ia and ib moving along the ring in both directions, since either the new jumps are very close to the old, or they are close to 0 and k .

case 2): trivial;

case 3): can be proved with arguments similar to case 1). ■

There are special graphs for which it is easy to calculate the optimal i even in the case $i > 1$. For instance we have the following:

Theorem 2.9 : *Given $k = sa + r$, with $0 < r < a$, $a < b$ and $a > \lfloor \frac{n}{k} \rfloor + 1$ ($k = s(k - a) + r$, with $0 < r < k - a$, $a > b$ and $b > \lfloor \frac{n}{k} \rfloor$ respectively), we have:*

1. *Greedy $_{\lfloor \frac{n}{k} \rfloor}$ is optimal for $a > \lfloor \frac{n}{k} \rfloor + 1$ ($b > \lfloor \frac{n}{k} \rfloor$), and for s odd and $0 < r \leq \lfloor \frac{n}{k} \rfloor$, or for s even and $0 < r \leq 2\lfloor \frac{n}{k} \rfloor + 1$ ($0 < r \leq 2\lfloor \frac{n}{k} \rfloor$).*
2. *Greedy $_{\lfloor \frac{n}{k} \rfloor}$ is optimal for $r > \lfloor \frac{n}{k} \rfloor$, and $a > \lfloor \frac{n}{k} \rfloor + 1$ ($b > \lfloor \frac{n}{k} \rfloor$), and for $a - \lfloor \frac{n}{k} \rfloor \leq r < a$ ($b - \lfloor \frac{n}{k} \rfloor \leq r < b$).*
3. *For $r > \lfloor \frac{n}{k} \rfloor$, $a - r \geq \lfloor \frac{n}{k} \rfloor + 1$ Greedy $_i$ is optimal for $i = \min \{i', i''\}$ and:*

(a) $(j - 1)\lfloor \frac{n}{k} \rfloor + j - 2 < r \leq j\lfloor \frac{n}{k} \rfloor + j - 1$, and if $j = 2d + 1$, $d = 0, 1, \dots$ then $i' = \lceil \frac{j-2}{2} \rceil + \lfloor \frac{s}{2} \rfloor$, else $i' = \lceil \frac{j-1}{2} \rceil + \lfloor \frac{s}{2} \rfloor$.

(b) $t\lfloor \frac{n}{k} \rfloor + t - 1 < a - r \leq (t + 1)\lfloor \frac{n}{k} \rfloor + t$, and if $t = 2d$, $d = 0, 1, \dots$ then $i'' = \lceil \frac{t+1}{2} \rceil + \lfloor \frac{s}{2} \rfloor$, else $i'' = \frac{t}{2} + \lfloor \frac{s}{2} \rfloor$.

and $j \leq s$ or $t \leq s$.

PROOF

1. It trivially follows from the fact that the remainder is small enough to be reached from a clockwise or a counter-clockwise direction.

2. Similar to 1).
3. Any further jump would take more steps then walking along the ring. $\lfloor \frac{r}{2} \rfloor$ steps are needed to move towards the middle, the other jumps are strictly related to the width of the remainder r . ■

Corollary 2.2 : *If $k = sa$, i.e., $(s - 1)a = b$, and $a > \lfloor \frac{n}{k} \rfloor + 1$, then $i = \lceil \frac{s-1}{2} \rceil$, $s \geq 1$.*

PROOF We have $a = (s - 1)b$, $2a = (s - 2)b, \dots, (s - 2)a = 2b$, $(s - 1)a = b$, that is the jumps in the counter-clockwise and in the clockwise direction coincide. The condition then is trivially implied. ■

2.4.4 Routing on Chordal Rings

In this section we show how *Greedy_i* is related to the Interval Labeling Scheme. We also present a routing function for the case of unidirectional and bidirectional chordal rings with one chord. There are cases in which the function computation is easy and requires $O(n \log n)$ bits and constant time. In the worst case the time complexity is at most $O(n)$ steps.

Definition 2.4 : *We denote with $i_c a$ the position closest to 0 that we reach jumping around the ring in a clockwise direction, with a number of step smaller then the distance along the ring. If the ring is unidirectional but the chords point in a counter-clockwise direction the symmetric point is $n - i_c a$.*

From the above definition we assume that the values a , $k - a$, etc. are positive, and are the ones computed in a unidirectional ring with chords oriented in a clockwise direction. The position $i_c a$ (see Notation 2.2 of section 2.4.1) can be calculated in linear time in the worst case, using the algorithm in Appendix. i_c is the number of jumps around the ring, m' is the distance from 0 to $i_c a$ moving with jumps of length k . (Note that $i_c a$ and i_c are respectively the s''' and the i' of the algorithm, whereas

m' is the same.) Note that $i_c \leq i$, where i is the optimal index (see Definition 2.3). The value of $n - i_c a$, m' , etc., can be derived from the previous model.

In the case of the unidirectional chordal ring a 1-ILS is given by the proof of the following theorem:

Theorem 2.10 : *There exists a 1-ILS for unidirectional chordal rings (and therefore an optimal BRS).*

PROOF Given a source node u we choose $[u, (u + i_c a) \bmod n]$ as the interval associated to the edge $e_1 = (u, (u + 1) \bmod n)$ and $[(u + i_c a) \bmod n, u]$ as the one associated to $e_2 = (u, (u + k) \bmod n)$. ■

The case of bidirectional chordal ring is more complicated. In [FGS94], it is shown that an optimal Interval Labeling Scheme (ILS) exists for the family of graphs for which $n \bmod k = 0$ (and consequently for $n \bmod n - k = 0$) and for which cyclic labels are used.

We now extend the previous result and we study a family of graphs for which ILS with cyclic labeling is optimal (that is a single interval for each edge is sufficient), and then we prove that this ILS is contained in *Greedy₀* in the case of bidirectional chordal rings.

Theorem 2.11 : *Given a bidirectional chordal ring, ILS with cyclic labeling is optimal if and only if $n \bmod k = 0$, $n \bmod n - k = 0$, or $n = sk + 1$, or $n = sk - 1$ and s is even, or $n = sk + 2$ and s is even for $n, s, k \in \mathcal{N}$.*

PROOF For $n \bmod k = 0$, $n \bmod n - k = 0$ see [FGS94]. Let us assume that $n = sk + 1$, or $n = sk - 1$ and s is even, or $n = sk + 2$ and s is even. If $n = sk + 1$ starting from 0 we can optimally reach the first $\lfloor \frac{k}{2} \rfloor$ nodes both in the clockwise and in the counter-clockwise direction. If s is even we are done. If s is odd, there are nodes that can be reached both from the edge $(0, k)$, than from $(0, n - k)$ with the same number of steps. We chose to reach them in such a way that to $(0, k)$ are associated with the nodes in $[\lfloor \frac{k}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor]$, and to $(0, n - k)$ the ones in

$[\lfloor \frac{n}{2} \rfloor + 1, n - \lfloor \frac{k}{2} \rfloor]$. The same holds for $n = sk - 1$ and s is even, or $n = sk + 2$ and s is even. For $n = sk$ see [FGS94].

Let us assume that we have an optimal cyclic labeling. We show that the chordal ring must be in one of the forms presented above. Let us assume $n = sk + r$, where $0 < r < k$. If $r = 2$ and s is odd, by moving $\lceil \frac{s}{2} \rceil$ times with steps of length k , we reach a point $> \lfloor \frac{n}{2} \rfloor$. This point can be reached in the counter-clockwise direction only with $\lceil \frac{s}{2} \rceil + 1$ moves. If $2 < r < k - 1$, this condition is even stronger. If $r = k - 1$ and s is odd with $\lceil \frac{s}{2} \rceil$ counter-clockwise moves we can reach a point $> \lfloor \frac{n}{2} \rfloor$ that can be reached in the counter-clockwise direction with an additional move. In this way we have covered all the cases. ■

We now show that in certain cases optimal ILS might not be obtained by cyclic labeling of the nodes, but by different labeling.

Corollary 2.3 : *There exist an infinite class of chordal rings for which an optimal ILS exists, but it is not cyclic.*

PROOF As an example consider for instance the chordal ring $C = (V, E', 3)$, with the labeling $V = \{0, 10, 20, 1, 11, 21, 2, 12, 22, \dots, 8, 18, 28, 9, 19\}$, and refer to the notation of Definition 2.2. An optimal ILS exists with the labeling $[6, 15]$ in $e_1 = (0, 1)$, $[1, 5]$ in $e_2 = (0, 3)$, $[24, 28]$ in $e_3 = (28, 0)$, $[16, 23]$ in $e_4 = (19, 0)$ in respect to the node 0.

On the other hand, if we give a cyclic ILS, there exists no single interval associated to edge e_2 (e_4 respectively) since nodes $0, \dots, 13$ and node 15 ($16, \dots, 28$, and 14) should be reached through e_2 (e_4) whereas node 14 (15) should be reached through e_4 (e_2).

We could trivially generalize this result by using bidirectional chordal rings that are not included in Theorem 2.11. For instance we could use the above labeling technique for all the chordal rings $C = (V, E', 3)$, with $n = sk - 1$ and s odd. ■

Theorem 2.12 : *ILS with cyclic labeling is contained in Greedy₀ for the case of a bidirectional chordal ring.*

PROOF We have to prove that where ILS with cyclic labels is defined, $i = 0$ for Greedy_i , and that the vice versa is not true.

If $n \bmod k = 0$ or $n \bmod n - k = 0$ after a whole walk around the graph, with chords of length k we end up in 0. This means that it is not convenient to go more then one time around the ring, so $i = 0$.

If $n = sk - 1$ and s is even or $n = sk + 1$ it is easy to see that $i = 0$ both for $k < \frac{n}{2}$, then for $k > \frac{n}{2}$. If $n = sk + 2$ and s is even the relation holds even for $s = 2$ since we consider only the nodes that are strictly included.

On the other hand, the example used in the corollary is a case for which ILS with cyclic labels does not work, but $i = 0$. ■

From the previous results we have seen that Greedy_0 is more powerful then ILS with cyclic labeling. We will show now how to use the results on Greedy_i to do routing on bidirectional chordal rings. We will see that in some cases we can define a simple boolean function that takes as input the source and destination labels, the values n and k and gives as output the outgoing edge on a shortest path to the destination. The function is very simple and the amount of memory required is $\leq 4 \log n$. Moreover, we will see that the routing we will define will cover all the cases in which Greedy_0 or Greedy_1 are defined.

W.l.o.g., we will analyze the case in which the source node is 0. We can extend these results to a generic source, by means of shifting the intervals.

For every chordal ring it is straightforward to calculate when to traverse edges e_1 or e_3 , but it is not simple to say when to take edges e_2 or e_4 (see Definition 2.2). The reason is that, if we build up a table that tells which edge to take for every destination, we will see that intervals e_2 and e_4 are alternating. This means that it is not true that to reach nodes in the interval $[0, \lfloor \frac{n}{2} \rfloor]$ we always have to take edge e_2 .

The method we have used to define the routing function is by constructing different tables and combining them together. In other words, we have defined four possible cases depending on which edge it is possible to take, that is:

Definition 2.5 : We define four different rings depending on which edge they contain, as:

Case I) R_1 : contains e_1 and e_2 (this is the above defined unidirectional case).

Case II) R_2 : contains e_4 and e_3 . (Symmetric to case 1. The points $n - a, n - 2a, \dots$ are reached this time moving in a counter-clockwise direction around the ring. These points are symmetric to $a, 2a, \dots$, with respect to node 0.)

Case III) R_3 : contains e_2 and e_3 .

Case IV) R_4 : contains e_1 and e_4 . (The points $n - a, n - 2a, \dots$ are the same as Case II.)

Each of these cases will be divided in two sub-cases, that is: i) $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$, and ii) $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$ for case I and II, or iii) $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$, and iv) $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ for case III and IV. The general case will be the combination of all these possible eight cases. We define very simple boolean functions in the case of chordal rings with bidirectional links of length 1 and the link of length k oriented in the clockwise or in the counter-clockwise direction. The case of $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$ and $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ will be solved only for particular chordal rings. (Later on we will present a different technique to solve this problem.) We will define the routing function using a constructive technique that, step by step, will combine more complicated tables.

Definition 2.6 : Given a destination node v , the routing function is a function $f: \{v\} \rightarrow \{e_1, e_2, e_3, e_4\}$.

Using the same notation of the unidirectional case (see Definition 2.4) we have the following:

Definition 2.7 : We denote by $j_c b$ (or $n - j_c b$) the position closest to 0 that we reach jumping around the ring with a number of steps smaller than the total number of steps of length 1 along the ring, in the opposite direction of the jumps.

We can extend the algorithm Index Search (see Appendix) to calculate $j_c a$ and its distance m'' .

Theorem 2.13 : *Given a chordal ring whose edges are defined in case I, II, III or IV, and given $i_c a$, $n - i_c a$, $j_c b$, and $n - j_c b$, we have the following routing function:*

1. *Case I: if $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ (i),*

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, k) \\ e_2, & \text{otherwise;} \end{cases}$$

if $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$ (ii) (Figure 2.3),

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, i_c a) \\ e_2, & \text{otherwise;} \end{cases}$$

2. *Case II: if $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ (i),*

$$f(v) = \begin{cases} e_4, & \text{for } v \in [0, n - k + 1) \\ e_3, & \text{otherwise;} \end{cases}$$

if $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$ (ii) (Figure 2.4),

$$f(v) = \begin{cases} e_4, & \text{for } v \in [0, n - i_c a + 1) \\ e_3, & \text{otherwise;} \end{cases}$$

3. *Case III: if $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$ (iii),*

$$f(v) = \begin{cases} e_2, & \text{for } v \in [0, \lfloor \frac{n}{k+1} \rfloor k + 1) \\ e_3, & \text{otherwise;} \end{cases}$$

if $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ (iv) (Figure 2.5),

$$f(v) = \begin{cases} e_2, & \text{for } v \in [0, n - j_c b + 1) \\ e_3, & \text{otherwise;} \end{cases}$$

Figure 2.3: Case I (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$, and $i_c = 1$.

Figure 2.4: Case II (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$, and $i_c = 1$.

4. Case IV: if $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$ (iii),

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, n - \lfloor \frac{n}{k+1} \rfloor k + 1) \\ e_4, & \text{otherwise;} \end{cases}$$

if $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ (iv) (Figure 2.6),

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, j_c b) \\ e_4, & \text{otherwise.} \end{cases}$$

PROOF The proof is straightforward and follows from Theorem 2.10. We will give just a sketch. If $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$, for 1 and 2, then is better to move along

Figure 2.5: Case III (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < k - a \leq k - 1$, and $j_c = 1$.

Figure 2.6: Case IV (ii) with $\lfloor \frac{n}{k} \rfloor + 1 < k - a \leq k - 1$, and $j_c = 1$.

the ring instead of jumping even one time around. If $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$, for 3 and 4, then is better to move along the ring instead of moving even less then one time around. The value $\lfloor \frac{n}{k+1} \rfloor k$ is a result of the relation: $\ell k \geq n - \ell$ that is $\ell \geq \lceil \frac{n}{k+1} \rceil$. All the other cases depend on the position of the jump $i_c a$ or $j_c b$. (Note that point a in Case 2 and 4 is symmetric, in respect to point 0, to point a in Case 1 and 3.) ■

We now combine cases 1 and 3 and cases 2 and 4. It is important to observe that the results are still valid independent from the fact that $a \leq k - a$ or $a > k - a$. We have now the following:

Theorem 2.14 : *Given a chordal ring whose edges are defined as case 1 and 3, or as case 2 and 4, we have the following routing function:*

1. Case 1 and 3:

(a) if $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ and $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, \lceil \frac{k}{2} \rceil + 1) \\ e_2, & \text{for } v \in [\lceil \frac{k}{2} \rceil + 1, n - \lfloor \frac{n}{k} \rfloor - \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor}{2} \rfloor) \\ e_3, & \text{otherwise;} \end{cases}$$

(b) if $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, \lceil \frac{k}{2} \rceil + 1) \\ e_2, & \text{for } v \in [\lceil \frac{k}{2} \rceil + 1, \lfloor \frac{n}{k+1} \rfloor k + \lfloor \frac{n-(k+1)\lfloor \frac{n}{k+1} \rfloor}{2} \rfloor + 1) \\ e_3, & \text{otherwise;} \end{cases}$$

(c) if $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$ and $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, i_c a - \lfloor \frac{i_c a - m'}{2} \rfloor + 1) \\ e_2, & \text{for } v \in [i_c a - \lfloor \frac{i_c a - m'}{2} \rfloor + 1, n - j_c b + \lfloor \frac{j_c b - m''}{2} \rfloor + 1) \\ e_3, & \text{otherwise.} \end{cases}$$

2. Case 2 and 4:

(a) if $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ and $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, k - a - \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor}{2} \rfloor + 1) \\ e_4, & \text{for } v \in [k - a - \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor}{2} \rfloor + 1, n - \lfloor \frac{k}{2} \rfloor + 1) \\ e_3, & \text{otherwise;} \end{cases}$$

(b) if $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, \lfloor \frac{n}{k+1} \rfloor + \lfloor \frac{n-(k+1)\lfloor \frac{n}{k+1} \rfloor}{2} \rfloor + 1) \\ e_4, & \text{for } v \in [n - \lfloor \frac{n}{k+1} \rfloor k - \lfloor \frac{n-(k+1)\lfloor \frac{n}{k+1} \rfloor}{2} \rfloor, n - \lfloor \frac{k}{2} \rfloor + 1) \\ e_3, & \text{otherwise;} \end{cases}$$

(c) if $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$ and $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, j_c b - \lfloor \frac{j_c b - m''}{2} \rfloor + 1) \\ e_4, & \text{for } v \in [j_c b - \lfloor \frac{j_c b - m''}{2} \rfloor + 1, n - i_c a - \lfloor \frac{i_c a - m'}{2} \rfloor + 1) \\ e_3, & \text{otherwise.} \end{cases}$$

PROOF 1) Case 1 and 3 (a): we observe that $j_c b = n - (k - a)$ since it is not worth to jump around the ring. We reach position $k - a$ with $\lfloor \frac{n}{k} \rfloor$ jumps of length k , but if we go one more time around we have a longer path than the one from $k - a$ and then along the ring, or from 0 along the ring in the counter-clockwise direction. Moreover, some of the nodes we have reached through e_1 in Case 1 can now be reached through e_2 moving then in the counter-clockwise direction along the ring. We will not prove all the other cases because they are a straightforward combination of the cases. We only observe that if $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$ the value of a is not important.

■

This last theorem is a proof of how easy is to define an ILS for the case of a ring whose links of length 1 are in both directions, and whose link of length k is either in the clockwise, or in the counter-clockwise direction.

At this point we can combine cases 1 and 3 with 2 and 4 of Theorem 2.14. We obtain a routing function that is totally defined but for the case $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k - 1$ and $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$. This case is solved only for $k = sa$, and $a \leq k - a$.

Since the chordal ring is symmetric, for the final boolean function defined in Theorem 2.15 we consider only the positive values of a and $k - a$. To reach the final results though we have to consider both the positive and the negative values.

Theorem 2.15 : Let $m_1 = \left\lfloor \frac{n-k-(k-1)\lfloor \frac{n}{k} \rfloor - 1}{2} \right\rfloor + 1$, and $m_2 = \left\lfloor \frac{(k+1)\lfloor \frac{n}{k} \rfloor - n}{2} \right\rfloor + 1$. Given a chordal ring whose edges are defined as case 1 and 3, and as case 2 and 4 of Theorem 2.14, we have the following routing function:

1. if $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ and $0 \leq k - a \leq \lfloor \frac{n}{k} \rfloor$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, \lfloor \frac{k}{2} \rfloor + 1) \\ e_3, & \text{for } v \in [n - \lfloor \frac{k}{2} \rfloor + 1, 0) \\ e_2, & \text{for } m \geq \lfloor \frac{n}{2k} \rfloor + 1 \text{ and } v \in [0, \lfloor \frac{n}{2} \rfloor + 1) \text{ or} \\ & \text{for } k\lfloor \frac{v}{k} \rfloor \leq \lfloor \frac{n}{2} \rfloor \text{ and} \\ & \lfloor \frac{v}{k} \rfloor \leq m - 1 \text{ and } v \geq \lfloor \frac{k}{2} \rfloor + 1 \text{ or} \\ & \lfloor \frac{v}{k} \rfloor k \leq v \leq \lfloor \frac{v}{k} \rfloor (k - 2) + \lfloor \frac{n}{k} \rfloor - \left\lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor + 2\lfloor \frac{v}{k} \rfloor}{2} \right\rfloor \text{ or} \\ & n - (\lfloor \frac{n}{k} \rfloor - \lfloor \frac{v}{k} \rfloor)k + \left\lfloor \frac{a-\lfloor \frac{n}{k} \rfloor + 2\lfloor \frac{v}{k} \rfloor + 1}{2} \right\rfloor + t < v \leq (\lfloor \frac{v}{k} \rfloor + 1)k \\ & \text{where } t = 0 \text{ if } \lfloor \frac{n}{k} \rfloor \text{ is odd, else } t = 2\lfloor \frac{v}{k} \rfloor + 1 - \lfloor \frac{n}{k} \rfloor, \text{ or} \\ & k\lfloor \frac{v}{k} \rfloor > \lfloor \frac{n}{2} \rfloor \text{ and } \lfloor \frac{n-v}{k} \rfloor + 1 \geq m \text{ and} \\ & v \geq n - (\lfloor \frac{n}{k} \rfloor - \lfloor \frac{v}{k} \rfloor)k + \left\lfloor \frac{a+\lfloor \frac{n}{k} \rfloor - 2\lfloor \frac{v}{k} \rfloor - 1}{2} \right\rfloor + t \text{ and} \\ & v \leq \lfloor \frac{v}{k} \rfloor k + \left\lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor - 2\lfloor \frac{v}{k} \rfloor}{2} \right\rfloor - 1 \\ e_4, & \text{otherwise;} \end{cases}$$

where $m = m_1$ if $a < k - a$, else $m = m_2$.

2. if $0 \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ and $\lfloor \frac{n}{k} \rfloor < k - a \leq k - 1$ (that is $a \leq k - a$), then

(a) For $0 \leq a \leq \lfloor \frac{n}{k} \rfloor - 1$, then $m = m_1$ and $f(v)$ as in 1.

(b) For $\lfloor \frac{n}{k} \rfloor \leq a \leq \lfloor \frac{n}{k} \rfloor + 1$ $m = 0$ and

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, \lfloor \frac{n}{k} \rfloor + \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor}{2} \rfloor + 1) \\ e_3, & \text{for } v \in [\lfloor \frac{n}{k} \rfloor k + \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor}{2} \rfloor, 0) \\ e_2, & \text{for } k\lfloor \frac{v}{k} \rfloor \leq \lfloor \frac{n}{2} \rfloor \text{ and} \\ & \lfloor \frac{v}{k} \rfloor k \leq v \leq \lfloor \frac{v}{k} \rfloor (k-2) + \lfloor \frac{n}{k} \rfloor + \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor + 2\lfloor \frac{v}{k} \rfloor}{2} \rfloor \text{ and} \\ & \lfloor \frac{v}{k} \rfloor \geq 1 \text{ or} \\ & n - (\lfloor \frac{n}{k} \rfloor - \lfloor \frac{v}{k} \rfloor)k + \lfloor \frac{a-\lfloor \frac{n}{k} \rfloor + 2\lfloor \frac{v}{k} \rfloor + 1}{2} \rfloor + t < v \leq (\lfloor \frac{v}{k} \rfloor + 1)k \\ & \text{where } t = 0 \text{ if } \lfloor \frac{n}{k} \rfloor \text{ is odd, else } t = 2\lfloor \frac{v}{k} \rfloor + 1 - \lfloor \frac{n}{k} \rfloor, \text{ or} \\ & k\lfloor \frac{v}{k} \rfloor > \lfloor \frac{n}{2} \rfloor \text{ and } v < \lfloor \frac{n}{k} \rfloor k + \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor - 2\lfloor \frac{v}{k} \rfloor}{2} \rfloor \text{ and} \\ & v \leq n - (\lfloor \frac{n}{k} \rfloor - \lfloor \frac{v}{k} \rfloor)k + \lfloor \frac{a+\lfloor \frac{n}{k} \rfloor - 2\lfloor \frac{v}{k} \rfloor - 1}{2} \rfloor + t \text{ and} \\ & v \leq \lfloor \frac{v}{k} \rfloor k + \lfloor \frac{k-a-\lfloor \frac{n}{k} \rfloor - 2\lfloor \frac{v}{k} \rfloor}{2} \rfloor - 1 \\ e_4, & \text{otherwise;} \end{cases}$$

3. if $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k-1$ and $0 \leq k-a \leq \lfloor \frac{n}{k} \rfloor$ (that is $k-a < a$), then $m = m_2$ and $f(v)$ as in 1.

4. if $\lfloor \frac{n}{k} \rfloor + 1 < a \leq k-1$ and $\lfloor \frac{n}{k} \rfloor < k-a \leq k-1$ then

(a) For $k = sa$, s integer, and $a \leq k - a$ then

$$f(v) = \begin{cases} e_1, & \text{for } v \in [0, \lfloor \frac{n}{k} \rfloor + \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor - 1}{2} \rfloor + 2) \\ e_3, & \text{for } v \in [n - \lfloor \frac{n}{k} \rfloor - 1 - \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor - 1}{2} \rfloor, 0) \\ e_2, & \text{for } v \in [\lfloor \frac{n}{k} \rfloor + \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor - 1}{2} \rfloor + 2, a \lfloor \frac{s-1}{2} \rfloor + 1) \text{ or} \\ & m = \lfloor \frac{v - a \lfloor \frac{s-1}{2} \rfloor}{k} \rfloor \text{ and} \\ & \text{for } v \leq \lfloor \frac{n}{2} \rfloor \text{ and} \\ & mk + a \lfloor \frac{s-1}{2} \rfloor \leq v \leq mk + a \lfloor \frac{s-1}{2} \rfloor + t \text{ where} \\ & t = \lfloor \frac{s-1}{2} \rfloor - m \text{ for } s \text{ odd or} \\ & t = \lfloor \frac{n}{k} \rfloor + \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor}{2} \rfloor - m \text{ for } s \text{ even or} \\ & v \geq (m+1)k - a + \lfloor \frac{n}{k} \rfloor - 1 - \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor + 1}{2} \rfloor + m \text{ and} \\ & v \leq (m+1)k + a \lfloor \frac{s-1}{2} \rfloor \text{ or} \\ & \text{for } \lfloor \frac{n}{2} \rfloor < v < n - \lfloor \frac{n}{k} \rfloor - 1 - \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor - 1}{2} \rfloor \text{ and} \\ & m = \lfloor \frac{n - v - a \lfloor \frac{s-1}{2} \rfloor}{k} \rfloor \text{ and} \\ & v \geq n - (m+1)k + \lfloor \frac{n}{k} \rfloor - 1 + \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor + 1}{2} \rfloor - m \text{ and} \\ & v \leq n - mk - a \lfloor \frac{s-1}{2} \rfloor - t \text{ where} \\ & t = \lfloor \frac{s-1}{2} \rfloor - m \text{ for } s \text{ odd or} \\ & t = \lfloor \frac{n}{k} \rfloor + \lfloor \frac{a - \lfloor \frac{n}{k} \rfloor}{2} \rfloor - m \text{ for } s \text{ even or} \\ e_4, & \text{otherwise.} \end{cases}$$

(b) For $k = s(k - a)$, s integer, and $a > k - a$ then $f(v)$ similar to (a), but intervals e_2 and e_4 are inverted and wider.

PROOF Case 1. The intervals associated with e_1 and e_2 are straightforward. For e_2 and e_4 it is not simple since depending on the chordal ring in some cases it is better to traverse edge e_2 , in other cases e_4 . We define m as the minimum number of jumps of length k for which there is an intersection between the intervals defined by e_2 and e_4 . The value m_1 is a result of the relation: $\lfloor \frac{n}{k} \rfloor - m_1 < m_1 + a + 1$. On the other hand, m_2 is given by the relation: $\lfloor \frac{n}{k} \rfloor - m_2 < m_2 - 1 + k - a + 1$. One of the two relations is weaker, depending upon the fact that $a < k - a$ or $a \geq k - a$.

The relation for the intervals is found in the same way m is defined. For instance, in Case 1, $m \geq \lfloor \frac{n}{2k} \rfloor + 1$ means that there is no alternation of intervals (i.e., there

exist an ILS). Moreover, the intervals are different if the destination is in the first ($k \lfloor \frac{v}{k} \rfloor \leq \lfloor \frac{n}{2} \rfloor$) or in the second half of the ring ($k \lfloor \frac{v}{k} \rfloor > \lfloor \frac{n}{2} \rfloor$). In the first half if $\lfloor \frac{v}{k} \rfloor \leq m - 1$ we are sure that v falls in an interval covered by e_2 , otherwise there are different intervals that can be derived considering the number of moves we make both in a clockwise and in a counter-clockwise direction. We find the closest spot in both directions (one is reached from e_2 , the other from e_4), and then we move along the ring. If $k \lfloor \frac{v}{k} \rfloor \leq \lfloor \frac{n}{2} \rfloor$ the destination v is reached from e_2 if it falls in one of the two subintervals that can be defined in $[k \lfloor \frac{v}{k} \rfloor, (k + 1) \lfloor \frac{v}{k} \rfloor]$. Moreover, one of the intervals might cross $\frac{n}{2}$. We can see this from $t = 0$. In the second half there is instead only one interval, or none if $\lfloor \frac{n-v}{k} \rfloor + 1 < m$. The function does not change if $a \leq k - a$, or $a > k - a$.

The other cases are similar to this. Case 2(a) is as 1, whereas in 2(b) the intervals previously defined only by e_1 and e_3 are now interleaved by e_4 and e_2 . Note that $f(v)$ for case 4(b) is straightforward to derive from case 4(a). ■

Corollary 2.4 : *Greedy₀ coincides with case 1, 2(a), and 3 of Theorem 2.15, case 2(b) and case 4 for $a = k - a$ (of the same theorem), are contained in Greedy₁.*

PROOF The proof for Greedy₀ follows directly from the definition given in the previous section (see Corollary 2.1). Case 2(b) and case 4 for $a = k - a$ are contained in Theorem 2.8. ■

Using the same technique as the one above defined, it is possible to define a boolean routing function for all the other chordal rings defined by Greedy₁ in case 1 and 3 of Theorem 2.15. We will not get into details, but again intervals of edges e_2 and e_4 are alternating and their number is limited since we cannot jump more than one time around the ring by definition of Greedy₁. This means that it is possible to define a very simple boolean function for all the rings that are included in Greedy₀ and Greedy₁ and for special cases of Greedy_i (case 4 of Theorem 2.15).

Moreover, one other interesting consequence of this theorem is the fact that, where 1-ILS fails, this boolean function technique works. In other words, we have the following:

Corollary 2.5 : *There exist chordal ring for which a ILS with cyclic labels requires at least $\lfloor \frac{n}{k} \rfloor - 2m + 1$ intervals, but a trivial BRS exists. The ILS requires $O(n^2 \log n)$ bits in the worst case, whereas BRS requires only $O(n \log n)$ bits and $O(n)$ time.*

PROOF See for instance, case 1 of Theorem 2.15, for $m < \lfloor \frac{n}{2k} \rfloor + 1$. Intervals for edge e_2 interleave the one for e_4 . The corresponding BRS is the function itself. If $k = 1$, $m = \frac{n}{4}$, then ILS requires $\frac{n}{2} + 1$ intervals. The rest comes trivially. ■

At this point we want to give a technique that permits us to define a routing function even for those chordal rings that have not been included in Theorem 2.15, that is the ones that are contained in case 4. The time complexity of the algorithm, that is of the routing function, is $O(i)$ steps, where $i = \max \{i_1, i_2, i_3, i_4\}$ and i_1, i_2, i_3, i_4 are the optimal indices of the *Greedy* algorithm applied to the four different kinds of rings of Definition 2.5. The i_1, i_2, i_3, i_4 can be calculated at the very beginning (that is will not be included in the cost of the routing function), just by simple modification of the algorithm Index Search. (Note that i_1 is exactly the i found by the original Index Search algorithm.) The cost in time of all these procedures is again $O(n)$ steps in the worst case.

Since $i \leq k \leq \lfloor \frac{n}{2} \rfloor$, the cost in time of the routing function remains linear. Moreover, the total amount of memory required is $O(n \log n)$ bits, since in each node is required to memorize only the i_1, i_2, i_3, i_4 , the node label, n and k .

We will now give a sketch of the algorithm. The main idea is that if the shortest path, from a source to a destination node, uses length 1 edges in the clockwise direction, it will never use length 1 edges in the counter-clockwise direction (the same with edges of length k). In other words, the shortest path will be the minimum of the shortest path found in $R_1, \dots, R_4$. We assume that v is the destination node.

Algorithm 1

1. For $R_1, \dots, R_4$ find the shortest path to v as follows:

- (a) For R_1 start from node $\lfloor \frac{v}{k} \rfloor$, for R_2 start from node $\lfloor \frac{n-v}{k} \rfloor$, for R_3 start from node $\lceil \frac{v}{k} \rceil$, and for R_4 start from node $\lceil \frac{n-v}{k} \rceil$. Calculate the distance

from this position to the destination and memorize it with its edge.

- (b) Using $i_1, \dots, i_4$ as maximum number of steps, calculate the shortest path for $R_1, \dots, R_4$ from each of the starting position, to the destination. Do this jumping around the ring, and each time you fall in the interval, check if the distance to the destination is smaller then the one you had before. If yes memorize the position, the distance and the edge (just the first time, then it will still be the same). Stop even if the distance you have moved to here, plus $\lfloor \frac{n}{k} \rfloor$ (or $\lfloor \frac{n}{k} \rfloor + 1$ depending on the kind of jumps you are doing around the ring) is bigger then the optimal distance you have found until now.
 - (c) Calculate the shortest path in $R_1, \dots, R_4$ by considering the jumps to the starting position, plus the optimal distance found at the previous step.
2. Calculate the shortest path among the one found for $R_1, \dots, R_4$ and choose the edge associated with it.

In conclusion we have the following theorem:

Theorem 2.16 : *It is always possible to define a boolean routing function for bidirectional chordal rings. This function has a time complexity of $O(i)$ steps in the worst case. The space required is $O(n \log n)$ bits.*

Chapter 3

Computing Boolean Functions

In this chapter we analyze the problem of computing boolean functions on an anonymous, asynchronous ring of n processors. The functions are computed on some initial data (each processor receives one bit and knows the size of the ring), and the output is the same for each processor. The complexity of a function depends on the number of bits exchanged during the computation. We introduce a function that requires $\Theta(n \log n)$ bits in almost every case. The function solves the string recognition problem, i.e., the problem of recognizing whether a string is equal, up to rotation, to the string that is input, bit by bit, to each processor.

The chapter is organized as follows. In Section 3.1 we introduce the problem. In Section 3.2 we summarize the results of this chapter. In Section 3.3 we define a preliminary notation. In Section 3.4.1 we introduce an upper bound for the string recognition problem in the case of periodic strings, and in Section 3.4.2 in the case of Kolmogorov random strings. Finally, in Section 3.5 we present the lower bounds of the problem.

3.1 Introduction

There have been several studies in the literature concerning the construction of efficient algorithms for computing functions on anonymous rings [ASW88, BMW94, FMR94, GS91, AM87, MW86], as well as on more general anonymous networks,

such as tori [BB89], hypercubes [KK91], Cayley networks [KK92], etc. In general, studies of the bit complexity of computing boolean functions mainly resort to input collection (i.e., the collection of different input bits stored in different processors), before determining the output of the boolean function.

In particular, Attiya, Snir and Warmuth [ASW88] show that all the functions that are computable in an anonymous, asynchronous ring of processors, can be computed using $O(n^2)$ messages (one bit messages if the functions are boolean). Each processor collects global information by forwarding n times a one bit message to one of its neighbours. Then it computes the function. In [ASW88] it also proved that every algorithm that computes the minimum of all inputs (the *OR* function in the boolean case), requires $\Omega(n^2)$ messages, i.e., the bounds in this case are tight. Furthermore, in [ASW88] the authors show that almost all boolean functions on n variables, require $\Omega(n^2)$ bits to compute. On the other hand, Moran and Warmuth [MW86] give an $\Omega(n \log n)$ lower bound for computing any non-constant function. If the function to be computed is constant, then no message needs to be sent. They also present functions (derived from a recursive application of de Bruijn sequences) with a $O(n \log n)$ bit complexity.

We investigate the bit complexity of computing boolean functions on an anonymous ring. We concentrate on an interesting class of functions related to the problem of recognizing an input as being equivalent up to rotation to a fixed string.

Previous algorithms for this problem have been “global” and do not take into account “local” patterns occurring in the string. Such patterns may be repetitive or discriminating, and can be used to provide efficient algorithms for recognizing strings. In this chapter we give new upper and lower bounds on the bit complexity of string recognition, and we show how the related function exhibits a behavior opposite to what was described above, in the case of the other functions. That is, for the case of a randomly chosen string (i.e., for almost all strings), an optimal algorithm that requires only $\Theta(n \log n)$ bits is given. It is interesting to note how Kolmogorov complexity theory is used in the following sections, in the proof of upper bounds, rather than its traditional application to the proof of lower bounds. For the

case of periodic strings, bounds are given which depend on the period of the string.

3.2 Results of the Chapter

Let $x = x_1x_2\cdots x_n, y = y_1y_2\cdots y_n \in \{0,1\}^n$ be binary strings of length n and suppose that for each $i = 1, \dots, n$ the i -th processor of an anonymous ring knows the i -th bit y_i of y as well as the entire string x . Give an efficient algorithm such that all n processors recognize whether or not the input string y is identical (up to rotation) to the string x is known as the string recognition problem.

We can reformulate our problem in terms of boolean functions. For any string $x \in \{0,1\}^n$ define the boolean function $f_x : \{0,1\}^n \rightarrow \{0,1\}$ as follows:

$$f_x(y) = \begin{cases} 1 & \text{if } y \text{ is a cyclic shift of } x \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, being invariant under rotations, the function f_x is computable in the oriented anonymous ring [ASW88]. We are interested in determining the bit complexity of computing $f_x(y)$.

It is well-known that every n -ary boolean function f which is computable on the anonymous oriented ring is a disjunction $\bigvee_{x \in E} f_x$ of some set of functions f_x , where x ranges over a finite set $E \subseteq \{0,1\}^n$. It follows that our string recognition problem is a special case of the well-known computability problem for boolean functions (on anonymous rings).

The input collection algorithm of Attiya, Snir and Warmuth [ASW88] implies that for every x , $f_x(y)$ can be computed using $O(n^2)$ bits. Such an algorithm is “global”, in the sense that the same algorithm is executed regardless of x or the input to the network. In Section 3.4.1, we show how taking into account repetitive or discriminating patterns within a string might provide more efficient algorithms for the problem. For the case of symmetric or repetitive strings we specify upper bounds (Section 3.4.1) and lower bounds (Section 3.5) which depend on the period k of a string x of length n , where $x = w^t v$, $t \geq 0$, v a prefix of w , and $k = |w|$. If

$x = w^i v, v = \emptyset$	$x = w^i v, v \neq \emptyset$	x is K -random
$O(nk + \frac{n^2}{k})$	$O(nk + n \log n)$	$O(n \log n)$
$\Omega(n \log n + \frac{n^2}{k})$	$\Omega(n \log n)$	$\Omega(n \log n)$

Table 3.1: Results of Chapter 3.

$v = \emptyset$ then we give an $O(nk + \frac{n^2}{k})$ upper bound and $\Omega(n \log n + \frac{n^2}{k})$ lower bound. If $v \neq \emptyset$ then we give an $O(nk + n \log n)$ upper bound and $\Omega(n \log n)$ lower bound.

Finally, we show (Section 3.4.2) that almost all strings can be recognized using $\Theta(n \log n)$ bits. Our upper bound is an interesting application of Kolmogorov complexity theory which has traditionally been used to show lower bounds. From an intuitive point of view, Kolmogorov complexity theory deals with the amount of information that a string contains [LV90, ZL70]. This quantity can be calculated in terms of the size of the shortest program that computes the string. For a random string x of length n , the length of its shortest program is very close to n , with high probability. A string with this property is called incompressible. We use this fact to derive an upper bound of $O(n \log n)$ bits for almost all strings, by showing that if no efficient algorithm exists to recognize a string then a short program exists to compute the string, contradicting its incompressibility.

Table 3.1 summarizes the bit complexity of recognizing a string x which is of the form $w^i v$, where $|w| = k$, k is the period of x and v is a proper prefix of w , and for a string x which is Kolmogorov random (see definition below).

3.3 Preliminaries

As our model of computation we consider the standard anonymous, asynchronous ring of n processors [ASW88]. The size n of the ring is known to all processors, and initially the processors are given a single bit as well as a string x of length n . The processors execute the same algorithm given the same data. At the end, it is

desired that all the processors determine correctly whether or not the given string is a rotation of the input string.

In the proofs below we further assume that the ring is oriented, i.e., the processors can distinguish in a “globally consistent” manner their left from their right neighbour. Nevertheless, all results are valid for unoriented rings with only minor modifications in the definitions and the statements of the results.

Let n be the number of processors and let $0 \leq i < n$ denote the i -th processor. If b_p is the input to processor p then $\langle b_p : p < n \rangle$ denotes the network input. The right (respectively, left) neighbourhood of processor p of length k is the sequence of bits $\langle b_{p+i} : 0 \leq i < k \rangle$ (respectively, $\langle b_{p-i} : 0 \leq i < k \rangle$), where the operations $+$, $-$ are done modulo n .

Before we proceed with the main algorithm we give some necessary definitions.

NOTATION 3.1 *We call u (respectively, v) a prefix (respectively, suffix) of the string x , if $x = uv$ for some string v (respectively, u); the prefix (respectively, suffix) is called proper if it is not equal to x . For any string x of length n let the period of x be the smallest integer $k \leq n$ such that for some strings w, v we have that*

- w has length k ,
- v is a prefix of w , and
- $x = w^{\lfloor n/k \rfloor} v$.

For any string u , let $|u|$ denote its length.

NOTATION 3.2 : *Given a string $x = x_1 x_2 \dots x_n$ of length n , let $x \upharpoonright s = x_1 x_2 \dots x_s$ denote the first s bits of x , and $x^r = x_{r+1} x_{r+2} \dots x_n x_1 x_2 \dots x_r$ the r -th rotation of x . S_x^t is the set of all the different substrings of x , and of all its rotations, that are of length $t \leq n$, i.e., $S_x^t = \{x^i \upharpoonright t \mid i = 1, \dots, n\}$.*

We will now give some preliminary definitions regarding Kolmogorov complexity theory. For further information regarding different applications of this theory the reader may refer to [LV90, ZL70].

Definition 3.1 : Given a string x of length n , and a program p of length $|p|$, a programming language S , let $K_S(x)$ denote the length of the minimum program that outputs x using S , i.e., $K_S(x) = \min\{|p| : S(p) = x\}$.

From now on we will use the notation $K(x)$ to refer to the Kolmogorov complexity of x with respect to an optimal method of specification S , i.e., a method whose complexity differs by at most a constant factor from other methods that compute x . If $K(x) \geq |x| - O(\log |x|)$ we say that the string x is loosely random, or $O(\log |x|)$ -incompressible, or Kolmogorov random. It is easy to derive the fact that almost every string of a given length is Kolmogorov random. To simplify our presentation we assume from now on that the string x is Kolmogorov random if $K(x) \geq |x| - \log |x|$. In particular, this implies that a string of length n is Kolmogorov random with probability at least $1 - 1/n$.

3.4 Upper Bounds

3.4.1 Periodic Strings

We now give an efficient algorithm for the string recognition problem. We can prove the following theorem.

Theorem 3.1 Let $x = w^{\lfloor n/k \rfloor} v$ be a string of period k such that $|x| = n$, $|w| = k$ and v is a proper prefix of w . There is an algorithm for recognizing the string x on an n processor anonymous oriented ring with bit complexity $O(nk + \frac{n^2}{k})$ if $v = \emptyset$, and $O(nk + n \log n)$ if $v \neq \emptyset$.

PROOF The main proof of the theorem consists of an efficient input collection algorithm satisfying the requirements above. Let a string x be known to all the processors and let b_p be the input to processor p . In the output of the algorithm it is determined whether or not x is a rotation of the input. Each processor executes the following algorithm which is described in several phases.

Phase 1

The processor computes the period k of the given string x as well as strings w, v such that v is a proper substring of w and in addition $x = w^t v$, for some integer $t \geq 0$. This is a local step executed initially by all the processors and does not contribute in any way to the overall bit complexity of the final algorithm.

Phase 2

Let k be the period of the string x as computed in phase 1. The processors now compute their left neighborhood of length k as well as their right neighborhood of length $2k + 1$ by executing the following algorithm.

```
/* computing the right neighborhood */
send  $b_p$  to the left;
for  $i = 1$  to  $2k$  do
    send bit received from the right to the left
od
/* computing the left neighborhood */
send  $b_p$  to the right;
for  $i = 1$  to  $k - 1$  do
    send bit received from the left to the right
od
```

The idea of this phase is to compute a *small* length neighborhood of the input string and use this to decide whether or not the input string is a rotation of x . It will become apparent in the case $v \neq \emptyset$ (discussed in the sequel) why we need to compute a neighborhood of length $2k + 1$ to the right and a neighborhood of length k to the left (for the time being we will only have use for the left neighborhood of length k). Clearly, the bit complexity of phase 2 is $O(nk)$.

Phase 3

In this phase the processors will execute the main part of the algorithm that will determine whether or not the given string is a rotation of the input. The rest of the algorithm depends on whether or not $v = \emptyset$.

CASE $v = \emptyset$:

Let u_p be the first k bits of the right neighborhood computed by processor p in phase 2 above. Now p executes the following algorithm.

```

if  $u_p$  is not a rotation of  $w$ 
  then broadcast "no"  $x$  is not a rotation of the input;
else /*  $u_p$  is a rotation of  $w$  */
  if  $u_p$  is the lexicographically maximal rotation of  $w$ 
    then send a "yes" around the ring;
    every processor that receives a "yes" forwards it
    around the ring;
    every processor that receiving a "no" forwards it
    around the ring and it stops;
  if  $u_p$  is the lexicographically maximal rotation of  $w$ 
    and processor  $p$  receives  $\lfloor \frac{n}{k} \rfloor$  "yes"
    then broadcast  $x$  is a rotation of the input;
    else /* it receives a "no" */
      broadcast  $x$  is not a rotation of the input;
fi

```

At the confirmation stage at most n/k processors participate (i.e., the ones that have a lexicographically maximal rotation of w); each processor sends a "yes" or a "no" around the ring (confirming that it has a lexicographically maximal rotation of w) and waits for $\lfloor \frac{n}{k} \rfloor$ "yes" or a no. At most $\lfloor \frac{n}{k} \rfloor$ "yes" are sent and travel n steps. Finally, if any processor either has a neighborhood of length k which is not

a rotation of w or receives a "no" message, it sends it to its neighbor and does not transmit again. Thus the overall bit complexity of this phase is $O(\frac{n^2}{k})$. (The "yes" or "no" messages require only $O(1)$ bits.)

At the end of this phase all processors will know whether or not all processors have seen a rotation of w . If indeed, all processors have seen a rotation of w then the string x is accepted else the string is rejected. The correctness of this last assertion is proved in the sequel.

In the following argument we assume that the strings x, w, v are as in the statement of Theorem 3.1.

If there exists a processor $p \in \{i = 1, \dots, n\}$ that does not see a rotation of the input, this obviously implies $x \neq w^t$, and p will broadcast a message to stop the other processors.

If all the processors have seen a rotation of the input, then $x = w^t$ must hold. Let us assume on the contrary that there exist a processor p whose left neighborhood $w^{(t-1)} = w_t \dots w_k w_1 \dots w_{t-1}$, $t = 1, \dots, k$ is a rotation of the input, but its neighbor $p + 1$ sees another rotation $w^{(s-1)} = w_s \dots w_k w_1 \dots w_{s-1}$, $s = 1, \dots, n$, which is not adjacent. Since p and $p + 1$ are adjacent, their left neighborhood must have $k - 2$ bits that are the same, i.e., $w_{(t+1) \bmod k} \dots w_k w_1 \dots w_{t-1} = w_s \dots w_k w_1 \dots w_{s-2}$. Moreover, if the two rotations cannot be adjacent, we have $w_t \neq w_{s-1}$, and since we are working with binary strings, we have $w_t = \overline{w_{s-1}}$. We now need the following:

Lemma 3.1 : *Every two rotations $w^{(t-1)} = w_t \dots w_k w_1 \dots w_{t-1}$, $t = 1, \dots, k$, and $w^{(s-1)} = w_s \dots w_k w_1 \dots w_{s-1}$, $s = 1, \dots, k$ of the binary string $w^{(0)} = w_1 \dots w_k$ of length k have the same number of ones.*

PROOF Let us assume by contradiction that the number of ones in $w^{(t-1)}$ and $w^{(s-1)}$ is different. By definition of rotation, if we start rotating both the bits of $w^{(t-1)}$ and of $w^{(s-1)}$ at a certain point we obtain the string $w^{(0)}$, and this implies that the number of ones must be the same. ■

We have assumed that there can be two rotations with a different number of zeros since $w_t \neq w_{s-1}$, but the above lemma shows that this is not possible, i.e., we

have a contradiction. If each processor sees a rotation of the input, then the string can only be $x = w^t$.

CASE $v \neq \emptyset$:

We assume that $x = w^t v$, where $v = v_1 \dots v_l$ is a proper prefix of w of length l , i.e., $v = w_1 \dots w_l$. Using the fact that w is the period of the string x , and assuming $x = x_1 \dots x_n$, we can prove easily the following lemma.

Lemma 3.2 : *There exists a substring $z = z_1 \dots z_k$ of length k of x that has weight (i.e., number of bits equal to 1) different from the weight of w .*

PROOF Easy by contradiction, using the fact w is the period of x , and that $v \neq \emptyset$.

■

Let v_p be the neighborhood of length $3k$ computed by processor p in phase 2. It consists of the left neighborhood of length $2k + 1$ and the right neighborhood of length k . In view of the representation of x in the form $w^t v$ it is an immediate consequence of Lemma 3.2 that there is a *unique* position in the string x whose k -neighborhood is not a rotation of w , but such that the k -neighborhoods of all k positions to its left are indeed rotations of w and the k positions to its right have the *correct* values.

However, the converse of this is also true, in the sense that if the input string has a unique position satisfying the above conditions, as well as the k positions to its right have the correct k -neighborhoods, and in addition all other positions have a k neighborhood which is a rotation of w then in fact the input string itself must be a rotation of x . In the proof of the following lemma we assume that $x = w^{\lfloor \frac{n}{k} \rfloor} v$, $\lfloor \frac{n}{k} \rfloor \geq 2$. The case $x = wv$ can be solved using input collection with $O(n^2)$ bit complexity.

Lemma 3.3 : *If after phase 2 of the algorithm there exists at least a processor whose $3k$ -bits string obtained in Phase 2 is different from $3k$ bits of the string $w^3 v w^3$, then*

$x \neq w^{\lfloor \frac{n}{2} \rfloor} v$. If on the other hand all the processors see a correct $3k$ -neighborhood, then $x = w^{\lfloor \frac{n}{2} \rfloor} v$, or $x = w^{s_1} v w^{s_2} v \dots w^{s_i} v$, for $s_j \geq 2$, $j = 1, \dots, i$, $i \geq 2$.

PROOF The first part of the proof is straightforward. For the second part let us then assume that the $3k$ -bits string is correct, and let us examine the $3k$ (correct) bits that two processor that are neighbors can see. We will denote them with $Y_1 = y_1 \dots y_{3k}$, and $Y_2 = y_{3k+1} \dots y_{6k}$. We will now show that the only strings that can occur are $x = w^{\lfloor \frac{n}{2} \rfloor} v$, or $x = w^{s_1} v w^{s_2} v \dots w^{s_i} v$, for $s_j \geq 2$, $j = 1, \dots, i$, $i \geq 2$. There are four cases:

1. Y_1 and Y_2 are both rotations of w^3 .

This implies that $y_1 = y_{3k+1}$ (same as the proof for the case $v = 0$).

2. Y_1 is a rotation of w^3 and Y_2 is a rotation of $3k$ bits of $w^3 v w^3$, but $Y_2 \neq w^3$.

Since $v_1 \dots v_l = w_1 \dots w_l$ and the strings Y_1 and Y_2 are consecutive, for $Y_1 = w_s \dots w_k w^2 w_1 \dots w_{s-1}$ we have:

If $s \leq k$ there are two cases. $Y_2 = w_{s+1} \dots w_k w^2 w_1 \dots w_{s-1} y_{3k+1}$ if $s < k$, $Y_2 = w^2 w_1 \dots w_{k-1} y_{3k+1}$ if $s = k$. This implies $y_{3k+1} \neq w_s$, because Y_2 is not a rotation of w^3 , so it must be $y_{3k+1} = w_1$ and $l = s - 1$. There are then $3k + 1$ consecutive bits in the resulting string, in the form $w_s \dots w_k w^2 v w_1$, $|v| = s - 1$.

3. Y_1 is a rotation of $3k$ bits of $w^3 v w^3$, but $Y_1 \neq w^3$, and Y_2 is a rotation of w^3 .

Since $v_1 \dots v_l = w_1 \dots w_l$ and Y_1 and Y_2 are consecutive strings, for $Y_2 = w_t \dots w_k w^2 w_1 \dots w_{t-1}$ we have:

- (a) If $t \geq 2$, then $Y_1 = y_1 w_t \dots w_k w^2 w_1 \dots w_{t-1}$, and $y_1 \neq w_{t-1}$ because Y_1 is not a rotation of w^3 . So it must be $y_1 = w_k \neq w_{t-1}$, and $w_t \dots w_k = w_1 \dots w_{k-t+1} = v_1 \dots v_l$, and $k - t + 1 = l$. If this is not true, the two neighbors cannot have seen a correct neighborhood. There are then $3k + 1$ consecutive bits in the form $w_1 w_t \dots w_k w^2 w_1 \dots w_{t-1} = w_1 v w^2 w_1 \dots w_{k-l}$, $|v| = k - t + 1$.

- (b) If $t = 1$, then $Y_1 = y_1 w^2 w_1 \dots w_{k-1}$, and $y_1 \neq w_k$ because Y_1 is not a rotation of w^3 , so it must be $y_1 = v_l \neq w_k$. There are then $3k + 1$ consecutive bits in the form $v_l w^2 w_1 \dots w_{k-1}$.

4. Both Y_1 and Y_2 are rotations of $3k$ bits of $w^3 v w^3$, but $Y_1, Y_2 \neq w^3$.

This case is much more complicated than the others since the substring v can be in different positions within the $3k$ bits. In most of the cases we will assume that Lemma 3.2 is true, i.e., by looking at k consecutive bits, we can always tell whether they are a rotation of w or not. We will refer to the following:

Lemma 3.4 *The strings $w = w^{(0)}, w^{(1)}, \dots, w^{(k-1)}$ are all different.*

PROOF If on the contrary any two of the above strings were equal then we could find a string u such that $u^r = w$, for some integer $r > 1$, thus contradicting the fact that w is the period of the string x . ■

- (a) $Y_1 = w_s \dots w_k w^2 v_1 \dots v_l w_1 \dots w_{s-l-1}$ (i.e., $w_1 \neq w_{l+1}$), $s \leq k$, $l < s - 1$.

This implies $Y_2 = w_{s+1} \dots w_k w^2 v_1 \dots v_l w_1 \dots w_{s-l-1} y_{3k+1}$ for $s < k$, or $Y_2 = w^2 v_1 \dots v_l w_1 \dots w_{k-l-1} y_{3k+1}$ for $s = k$. The only possible situation is $y_{3k+1} = w_{s-l}$, otherwise the processor don't see a correct configuration. The $3k$ consecutive bits are then $w_s \dots w_k w^2 v w_1 \dots w_{s-l}$, $s \leq k$.

- (b) i. $Y_1 = w_s \dots w_{l+1} \dots w_k w v w_1 \dots w_{k-l+s-1}$ (i.e., $w_1 \neq w_{l+1}$), $1 \leq s < l + 1 \leq k$. The only situation that can occur is then the one with

$Y_2 = w_{s+1} \dots w_k w v w_1 \dots w_{k+s-l-1} y_{3k+1}$ and $y_{3k+1} = w_{k-l+s}$. The $3k$ consecutive bits are then $w_s \dots w_k w v w_1 \dots w_{k-l+s}$.

- ii. $Y_1 = w_{l+1} \dots w_k w v w_1 \dots w_{k-l+s-1}$ (i.e., $w_1 \neq w_{l+1}$), $1 \leq s = l+1 < k$.

This implies $Y_2 = w_{l+2} \dots w_k w v w y_{3k+1}$. The only possible situation is $y_{3k+1} = w_1$ (Lemma 3.4), and the $3k$ consecutive bits are then $w_{l+1} \dots w_k w v w w_1$.

- iii. $Y_1 = w_s \dots w_k w v w w_1 \dots w_{s-l-1}$ (i.e., $w_1 \neq w_{l+1}$), $l + 1 < s \leq k$.

This implies $Y_2 = w_{s+1} \dots w_k w v w w_1 \dots w_{s-l-1} y_{3k+1}$ for $s < k$, $Y_2 =$

$wvw w_1 \dots w_{s-l-1} y_{3k+1}$ for $s = k$. The only situation is $y_{3k+1} = w_{s-l}$, otherwise the processors don't see a correct configuration. The $3k$ consecutive bits are then $w_s \dots w_k wvw w_1 \dots w_{s-l}$.

- (c) i. $Y_1 = w_s \dots w_{l+1} \dots w_k v_1 \dots v_l w w_1 \dots w_{k-l+s-1}$, $1 \leq s < l+1 \leq k$. This implies $Y_2 = w_{s+1} \dots w_k v w w_1 \dots w_{k-l+s-1} y_{3k+1}$. The only possible situation is $y_{3k+1} = w_{k-l+s}$, and the $3k$ consecutive bits are then $w_s \dots w_k v w w_1 \dots w_{k-l+s}$.
- ii. $Y_1 = w_{l+1} \dots w_k v_1 \dots v_l w^2$, $1 \leq s = l+1 \leq k$. Note that this is different from w^3 for Lemma 3.4. This implies $Y_2 = w_{l+2} \dots w_k v w^2 y_{3k+1}$. The only possible situation is $y_{3k+1} = w_1$. The $3k$ consecutive bits are then $w_{l+1} \dots w_k v w^2 w_1$.
- iii. $Y_1 = w_s \dots w_k v_1 \dots v_l w^2 w_1 \dots w_{s-l-1}$, $l+1 < s < k$. This implies that $Y_2 = w_{s+1} \dots w_k v w^2 w_1 \dots w_{s-l-1} y_{3k+1}$ for $s < k$, or that $Y_2 = v w^2 w_1 \dots w_{k-l-1} y_{3k+1}$ for $s = k$. I.e., if $w_{s+1} \dots w_k v = w_{k+l-s+1} \dots w_k$ for $s < k$ or $v = w_{k-l} \dots w_k$ for $s = k$, and $2l = s-1$, then $y_{3k+1} = w_1$, else $y_{3k+1} = w_{s-l}$. The $3k$ consecutive bits are then $w_s \dots w_k v w^2 v w_1$ in the first case, or $w_s \dots w_k v w^2 w_1 \dots w_{s-l}$ in the second.
- iv. $Y_1 = w_k v_1 \dots v_l w^2 w_1 \dots w_{k-l-1}$, $l+1 < s = k$. This implies $Y_2 = v_1 \dots v_l w^2 w_1 \dots w_{k-l-1} y_{3k+1}$. The only situation that can occur is if $v_1 \dots v_l = w_{k-l} \dots w_k$ and $k = 2l - 1$ then $y_{3k+1} = w_1$, else $y_{3k+1} = w_{k-l}$. The $3k$ consecutive bits are then $w_k v w^2 v w_1$ in the first case, and $w_k v w^2 w_1 \dots w_{k-l}$ in the second.
- v. $Y_1 = v_m \dots v_l w^2 w_1 \dots w_{k+m-l+1}$, $1 \leq m \leq l$. This implies $Y_2 = v_{m+1} \dots v_l w^2 w_1 \dots w_{k+m-l} y_{3k+1}$, i.e., if $v_{m+1} \dots v_k = w_{k+m-l-1} \dots w_k$, and $w_m \neq w_{k-l+m}$, and $2l = k + m$, then $y_{3k+1} = w_1$, else $y_{3k+1} = w_{k+m-l}$. The $3k$ consecutive bits are then $v_m \dots v_l w^2 w_1 \dots w_{k+m-l}$ in the first case, or $v_m \dots v_l w^2 w_1 \dots w_{k+m-l-1} w_1 = v_m \dots v_l w^2 v w_1$ in the second.

It can never happen that two strings v are close to each other less than $2k$ bits, i.e.,

of a string w^2 . This condition comes from the fact that, if we try to build up a string of length n starting from Y_1 and shifting one bit to the right at the time, the resulting string will maintain the above condition. This can be viewed as follows: if a new string $Y_3 = y_3 \dots y_{3k+3}$ is built starting from the string Y_2 we obtained before, we will fall again in one of the above cases. The only critical case is case 4 (c) iii), iv), and v), where the string v occurs twice while building Y_2 from Y_1 , but Y_2 sees only the rightmost one. Y_3 will anyway see the string v to the right of w^2 , i.e., Y_2 and Y_3 will fall again in case (4) a and so on. This implies that the only possible resulting strings are either $x = w^{\lfloor \frac{n}{2} \rfloor} v$, or $x = w^{s_1} v w^{s_2} v \dots w^{s_i} v$, for $s_j \geq 2$, $j = 1, \dots, i$, $i \geq 2$. ■

It is now easy to convert this characterization into an $O(nk + n \log n)$ algorithm for determining whether or not the input is a rotation of x . The algorithm for processor p is as follows.

```

if a proper rotation is not seen or a halt is received
  then broadcast  $x$  is not a rotation of the input and halt
  else if in the "unique" position
    then send a message around the ring with a counter  $c_1$ 
    fi;
    counter  $c_1$  is incremented by every processor;
    if in the "unique" position,
      and a counter  $c_1 < n$  is received,
        then broadcasts  $x$  is not a rotation of the input and halt
        else if  $c_1 = n$ ,
          then broadcast  $x$  is a rotation of the input;
          fi
        fi
    fi
fi

```

The observations made above prove the correctness of the algorithm. Only one processor can be in the unique position. If there is more than one, this situation

is detected by another processor in a unique position, by receiving $c_1 < n$. Note that it is not possible that every processor has w^3 as a neighborhood ($|x| = n$ cannot be divided by $|w|$, since $|v| < |w|$). The bit complexity of the algorithm is straightforward. In the worst case in this step, a message of at most $\log n$ bits travels for n steps. The total bit complexity of the algorithm, in the case $v \neq 0$ is then $O(nk + n \log n)$. This completes the proof of Theorem 3.1. ■

3.4.2 Kolmogorov Random Strings

We now present an algorithm that optimally finds using $\Theta(n \log n)$ bits, whether a Kolmogorov random string x is a rotation of the input checking only a logarithmic number of bits. (The lower bound follows from [MW86].) In particular this shows that almost all strings can be recognized using $\Theta(n \log n)$ bits.

Theorem 3.2 : *Let x be a Kolmogorov random string of length n . There is an algorithm for recognizing the string x on an n processor anonymous oriented ring with bit complexity $O(n \log n)$.*

PROOF Let x be a Kolmogorov random string of length n , given as an input to all the processors, and let b_p be a single bit of input at processor p . In the output of the algorithm it is determined whether or not x is a rotation of the input configuration. In the algorithm we use a constant c that will be more precisely calculated in Lemma 3.5. We will refer to Lemma 3.5 and 3.6 to justify the fact that, if the $c \lceil \log n \rceil$ neighborhood of a processor coincides with one of the strings contained in $S_x^t = \{x^i \uparrow t \mid i = 1, \dots, n\}$ (see Section 3.3), then, for n large enough, the input string must be x or one of its rotations.

Let $c = \max\{c', c''\}$, where c' and c'' are constants given in Lemma 3.5 and 3.6.

A generic processor p executes the following phases.

Phase 1

compute and store $S_x^t = \{x^i \uparrow t \mid i = 1, \dots, n\}$, for $t = c \log n$. This is a local step and it does not contribute on the overall bit complexity of the final algorithm.

Phase 2

compute the right neighborhood of length t .

*/*computing the right neighborhood*/*

send b_p to the left;

for $i = 1$ to t do

 send bit received from the right to the left;

 od check whether the t -neighborhood coincides with one of the strings contained in $S_x^t = \{x^i \uparrow t \mid i = 1, \dots, n\}$. The cost of this phase, in terms of bits transmitted, is $O(n \log n)$.

Phase 3

In this phase p will determine if the input string is a rotation of x .

Since x is a Kolmogorov random string, if after $t - 1$ steps all the processors have seen strings of $S_x^t = \{x^i \uparrow t \mid i = 1, \dots, n\}$, then the input string must be x (see Lemma 3.5), otherwise, if at least one finds a different substring, it is not x .

Let u_p be the t -neighborhood of processor p . Now p executes the following:

if $u_p \notin S_x^t = \{x^i \uparrow t \mid i = 1, \dots, n\}$,

 then broadcast x is not a rotation of the input and halt;

else if u_p is the maximal string

 then send a counter c_0 around the ring;

/ counter c_0 is only incremented by processors who have seen an acceptable string */*

 if a counter value $c_0 = n$ is received

 then broadcast x is a rotation of the input and halt;

 else */* $c_0 < n$, or another processor*

 has seen a not acceptable string and has sent a halt */

 broadcasts x is not a rotation of the input and halt

fi

```

else /*  $u_p$  is not a maximal string */
if a not acceptable string is seen
then broadcast  $x$  is not a rotation of the
input and halt
else if a counter  $c_0$  is received
then broadcast  $c_0 := c_0 + 1$ 
else /* a halt is received */
store the information " $x$  is (is not) a
rotation of the input", broadcast it and halt
fi
fi
fi
fi
fi

```

We can now precisely define a method to find the right value of c and we can justify phase 2 and 3 of the algorithm.

Definition 3.2 : Two strings $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_n$ of length n , a substring of y^i $s = s_1 \dots s_t = a_2 \dots a_tv_1$, of length $t < n$, $1 \leq i \leq n$, and a substring $a_1 \dots a_tu_1$ of x^j , $1 \leq j \leq n$ are given. We define as right overlap (respectively left overlap) of length $l \geq 1$ of $a = a_1 \dots a_t$ in x^j and s , the substring $a_{t-l+1} \dots a_t$ (respectively $a_1 \dots a_l$) of a that coincides with the substring $s_1 \dots s_l$ (respectively $s_{t-l+1} \dots s_t$) of s . If $l = 0$ we say that there is no overlap.

We have the following:

Lemma 3.5 : There exist a constant $c' \in \mathcal{N}$, such that, for n sufficiently large and for all $t \geq c' \log n$, for any two strings $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_n$ of length n , if x is Kolmogorov random, and $S_y^t \subseteq S_x^t$, then $x = y$ up to rotation.

Figure 3.1: The right overlap is of length $0 < l < \frac{t}{2}$, and $n \geq 1 + 2(t - l - 1)$

PROOF The proof of the lemma follows from results reported in [S86]. We extend and present in detail the method of [S86], analyzing all different cases.

Given an integer $t < n$, let $a \in S_y^t$, i.e., $y^i = a \cdot v = a_1 \dots a_t v_1 \dots v_{n-t}$ for some $i \in \{1, \dots, n\}$ and some string v , $|v| = n - t \geq 1$. Since $a \in S_x^t$, it is clear that $x^j = a \cdot u = a_1 \dots a_t u_1 \dots u_{n-t}$ for some $j \in \{1, \dots, n\}$ and some string u , $|u| = n - t \geq 1$. Consider the substring s of y^i , $s = s_1 \dots s_t = a_2 \dots a_t v_1$ of length t , $s \in S_y^t \subseteq S_x^t$. The proof of the lemma is divided in different cases depending on the different overlaps of the strings s and a in x^j . W.l.o.g we assume that the overlap does not coincide with the bits $a_2 \dots a_t u_1$ since, if it is the case, we can always rotate the string x^j up to a point in which it does not coincide, or otherwise we easily obtain the condition $x = y$ up to rotation.

Let l be the length of the overlap of s in x^j .

- Case 1): There is a right overlap of a in x^j and s , of length $0 < l < \frac{t}{2}$. Moreover, $n \geq 1 + 2(t - l - 1)$. (See Figure 3.1.)

The overlap covers the bits $a_{t-l+1} \dots a_t$. There might even be a left overlap on a , but w.l.o.g. we will not consider this condition (since, if it is the case, we are able to compress more the coding of the string x). Let us assume $s_1 = a_{t-l+1} = a_2, \dots, s_{l-1} = a_t = a_l, \dots, s_l = u_1 = a_{l+1}, \dots, s_t = u_{t-l} = v_1$. Hence the substring $a_2 \dots a_{t-l}$ of length $t - l - 1 \geq \frac{t}{2}$ is repeated at least twice in x^j .

We now prove that $K(x) < n - \log n$ which contradicts the incompressibility of x . We encode x as a_1 plus twice a string of length $t - l$, plus the remaining string x' of length $n - 2(t - l) - 1$. Hence x is given first by specifying a_1 in $O(1)$. The encoding of the two strings $a_2 \dots a_{t-l}$ is as follows: let T be a

Figure 3.2: The right overlap is of length $0 < l < \frac{t}{2}$, and $n < 1 + 2(t - l - 1)$

Turing Machine that computes $a_2 \dots a_{t-l}$ from a program p . Let $\text{num}(T) = n$ be the number associated with the Turing Machine T and fix a Universal Turing Machine V such that V started on the input $0^{\text{num}(T)}1p$ simulates T on input p and then doubles the output starting from position 2. If we start the reference Universal Turing Machine U on input $0^{\text{num}(V)}10^{\text{num}(T)}1p$ we will have the double string as output, starting from position 2. This will add a term $K(a_2 \dots a_{t-l}) + O(1)$ for the encoding of twice $a_2 \dots a_{t-l}$. We have a term of $O(\log 2(t - l) - 1)$ to remember the position of the end of the double string plus $O(\log j)$ bits for the index j of the rotation of the string x^j . The string $x' = u_{t-l+1} \dots u_{n-t}$ has length $n - 2(t - l) - 1$.

We have then:

$$\begin{aligned}
 K(x) &\leq O(1) + K(a_2 \dots a_{t-l}) + O(1) + O(\log(2(t - l) - 1)) \\
 &\quad + O(\log j) + K(x') \\
 &\leq t - l - 1 + O(\log n) + n - 2(t - l) - 1 \\
 &= n - (t - l) + O(\log n) < n - \frac{t}{2} + c_1 \log n,
 \end{aligned}$$

i.e., there exists a constant $c_1 \in \mathcal{N}$, such that, for large enough n , for $t > 2(c_1 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

- Case 2): There is a right overlap of a in x^j and s , of length $0 < l < \frac{t}{2}$. Moreover, $n < 1 + 2(t - l - 1)$, i.e., there is even a left overlap. (See Figure 3.2.)

The right overlap covers the bits $a_{t-l+1} \dots a_t$. There are two cases depending on the length of the substring of s that covers the last bits of x^j from position a_{t-l+1} , up to u_{n-t} (since the overlap is wrapping around). To be more precise

Figure 3.3: The left overlap is of length $0 < l < \frac{t}{2}$, and $n > 2t - l + 1$

let $m = l + n - t < t - l - 1$ be the length of this substring. A possible coding is given by repeating twice the last m bits of x^j . We have to remember the positions $m + 1 = l = n - t + 1$ and $t - l + 1$ where respectively the first of these repeated strings ends and the where the other starts. The encoding is given again by running the Turing Machine T on input p' for which it computes the string $a_2 \dots a_{m+1}$. The Universal Turing Machine V will repeat the string twice, starting from a_2 and a_{t-l+1} . We have then

$$\begin{aligned}
 K(n) &\leq O(1) + m + K(a_{m+1} \dots a_{t-l}) + O(\log(m + 1)) \\
 &\quad + O(\log t - l + 1) + O(\log j) \\
 &= O(1) + m + t - l - m + O(\log n) \\
 &= t - l + c_2 \log n,
 \end{aligned}$$

i.e., there exists a constant $c_2 \in \mathcal{N}$, such that, for large enough n , for $t - l < n - (c_2 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

Using this coding the condition $t - l < n - (c_2 + 1) \log n$, is sufficient. If $t - l \geq n - (c_2 + 1) \log n$ (i.e., $m = n - t + l < (c_2 + 1) \log n$), then $t > n - (c_2 + 1) \log n$, i.e., there is a right overlap of length $t - m > n - 2(c_2 + 1) \log n$. For large enough n , $n - 2(c_2 + 1) \log n > \frac{t}{2} = \frac{n - (c_2 + 1) \log n}{2}$, i.e., we fall in Case 5.

- **Case 3):** There is no right overlap and a left overlap of a in x^j and s , of length $0 \leq l < \frac{t}{2}$. (i.e., if $l = 0$ there is no overlap at all and $a_{t+1} = v_1$ in the figure.) (See Figure 3.3.)

The proof is done in the same way as the first subcase of case 2.

Figure 3.4: The right overlap is of length $l \geq \frac{t}{2}$

- Case 4): There is a right overlap of a in x^j and s , of length $l \geq \frac{t}{2}$. (See Figure 3.4.)

The right overlap covers bits $a_{t-l+1} \dots a_t$. We first observe that since we have assumed that $v_1 \neq u_1$, the overlap must contain at least two elements. This implies that the substring $a_2 \dots a_{t-l}$ is repeated at least $\lfloor \frac{t-1}{t-l-1} \rfloor + 1$ times. (We can encode this with $O(\log(\lfloor \frac{t-1}{t-l-1} \rfloor + 1))$ bits.) The encoding is given by a_1 , plus $\lfloor \frac{t-1}{t-l-1} \rfloor + 1$ times $a_2 \dots a_{t-l}$, plus a string x' of $n-1 - ((\lfloor \frac{t-1}{t-l-1} \rfloor + 1)(t-l-1))$ bits, plus the starting positions.

$$\begin{aligned} K(x) &\leq O(\log n) + t - l - 1 - (t - l - 1) + n - 1 \\ &\quad - \left(\left(\left\lfloor \frac{t-1}{t-l-1} \right\rfloor \right) (t-l-1) \right) \\ &\leq n + O(\log n) - \frac{l}{2}. \end{aligned}$$

Since $l \geq \frac{t}{2}$, there exists a constant $c_4 \in \mathcal{N}$, such that, for large enough n , for $t > 4(c_4 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

- Case 5): There is a left overlap of a in x^j and s , of length $l \geq \frac{t}{2}$. There might be or not a right overlap. (See Figure 3.5 for the case of the right overlap.)

The overlap covers the bits $a_1 \dots a_l$. This means that the last $t-l$ bits are $a_2 \dots a_{t-l+1}$. If $t < n - (t-l)$ and $(t-l+1) \geq \lfloor \frac{t}{2} \rfloor$ (i.e., the string $a_1 \dots a_{t-l+1}$ is repeated once in $a_1 \dots a_t$), or $t \geq n - (t-l)$ and $(t-l+1) \geq \lfloor \frac{n-(t-l)}{2} \rfloor$ (i.e., the string $a_1 \dots a_{t-l+1}$ is repeated once in $a_1 \dots a_{n-(t-l)}$), then the encoding is given by a_1 , plus $a_2 \dots a_{t-l+1}$ in positions 2 and $n - (t-l) + 1$, that is

$$K(x) \leq O(1) + n - (t-l) + O(\log n)$$

Figure 3.5: The left overlap is of length $l \geq \frac{t}{2}$

$$\leq n - \lfloor \frac{t}{2} \rfloor - 1 + c_5 \log n,$$

i.e., there exists a constant $c_5 \in \mathcal{N}$, such that, for large enough n , for $t > 2(c_5 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

If $t < n - (t - l)$ and $(t - l + 1) < \lfloor \frac{t}{2} \rfloor$ (i.e., the string $a_1 \dots a_{t-l+1}$ is repeated $\lfloor \frac{t}{t-l+1} \rfloor$ times in $a_1 \dots a_t$), or $t \geq n - (t - l)$ and $(t - l + 1) < \lfloor \frac{n-(t-l)}{2} \rfloor$ (i.e., the string $a_1 \dots a_{t-l+1}$ is repeated $\lfloor \frac{n-(t-l)}{t-l+1} \rfloor$ times in $a_1 \dots a_{n-(t-l)}$), then the encoding is given by $\lfloor \frac{t}{t-l+1} \rfloor$ (respectively $\lfloor \frac{n-(t-l)}{t-l+1} \rfloor$) times the string $a_1 a_2 \dots a_{t-l+1}$ plus the remaining substring. With some calculations we obtain:

$$K(x) \leq n - \frac{t}{2} + c_6 \log n,$$

i.e., there exists a constant $c_6 \in \mathcal{N}$, such that, for large enough n , for $t > 2(c_6 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction, or

$$K(x) \leq n - \frac{n - (t - l)}{2} + O(\log n) \leq \frac{n}{2} + c_7 \log n,$$

i.e., there exists a constant $c_7 \in \mathcal{N}$, such that, for large enough n , $K(x) < n - \log n$ which is a contradiction.

The constant c' of the statement of the lemma is then $c' = \max\{c_1, \dots, c_7\}$. This ends the proof of the Lemma 3.5. ■

To prove the correctness of phase 3 we need another lemma which ensures that, if a string is Kolmogorov random, then there must exist a unique maximal substring:

Figure 3.6: At least one overlap is of length $l > \frac{t}{2}$

Lemma 3.6 : *Given a string $x = x_1x_2 \dots x_n$ of length n , and an integer $t \leq n$, there exist a constant $c' \in \mathcal{N}$, such that, if n is large enough, for each $t \geq c' \log n$, if $|S_x^t| < n$, then x is not Kolmogorov random.*

PROOF Let us take two substrings s and s' of x which are equal. The proof is very similar to the one of Lemma 3.5. There are two cases:

- Case 1): The (right or left) overlap l of s and s' in x is of length $> \frac{t}{2}$. Rotate the string x in such a way that the corresponding x^j starts with one of the substrings (w.l.o.g. s), and has a right overlap of length $> \frac{t}{2}$. (There might even be a left overlap, but we will not consider it.) (See Figure 3.6.)

Since $t - l < \frac{t}{2}$, the string $s_1 \dots s_{t-l}$ is repeated at least twice, more precisely $\lfloor \frac{t}{t-l} \rfloor$ times. The coding can then be: repeat $\lfloor \frac{t}{t-l} \rfloor$ times the string $s_1 \dots s_{t-l}$ and add the final string. This leads to:

$$\begin{aligned} K(x) &\leq n - \left(\left\lfloor \frac{t}{t-l} \right\rfloor - 1 \right) (t-l) + O(\log n) \\ &\leq n - \frac{t}{3} + c_3 \log n, \end{aligned}$$

i.e., there exists a constant $c_3 \in \mathcal{N}$, such that, for large enough n , for $t > 3(c_3 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

- Case 2): Both the right and left overlap of s and s' in x are of length $\leq \frac{t}{2}$. Call l the biggest of the two and, w.l.o.g. assume that is the right overlap. Rotate the string x in such a way that the corresponding x^j starts with one of the substrings (w.l.o.g. s) and has l as a right overlap. (See Figure 3.7.)

Figure 3.7: Both overlaps are of length $l \leq \frac{t}{2}$

Since the right overlap is of length at most l , and the left overlap is $\leq l$, there are at least $t - l$ bits of s' that overlap the last bits of x^j . The encoding for $l > 0$, can be given as repeating twice the string $s_1 \dots s_{t-l}$ from position 1 and $t - l + 1$ for $l > 0$. This implies:

$$\begin{aligned} K(x) &\leq n - (t - l) + O(\log n) \\ &\leq n - \frac{t}{2} + c_9 \log n, \end{aligned}$$

i.e., there exists a constant $c_9 \in \mathcal{N}$, such that, for large enough n , for $t > 2(c_9 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

If $l = 0$, i.e., there is no right or left overlap, the entire string s is repeated twice, that is:

$$K(x) \leq n - t + c_{10} \log n,$$

i.e., there exists a constant $c_{10} \in \mathcal{N}$, such that, for large enough n , for $t > (c_{10} + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

The constant c'' of the statement of the lemma is then $c'' = \max\{c_8, c_9, c_{10}\}$. This ends the proof of Lemma 12. ■

If the input string is x then for Lemma 3.6 there exists a unique leader since only one processor sees a maximal substring. The bit complexity of this step is then $O(\sum_{i=0}^{\log n} 2^i(i+1)) + O_1(n) = O(n \log n)$. If the input string is not x , then there are at most $\frac{n}{c \log n}$, $c \in \mathcal{N}$, that see the maximal string and that send a counter. In this case

the bit complexity is still $O(n \log n) + O(n) = O(n \log n)$ since each counter will stop after reaching one of these processors, i.e., at most $c \log n$ bits message will travel for n steps. The correctness of this step is straightforward. If the input string is x , then there is a unique leader that acknowledges the fact that each processor has seen an acceptable neighborhood, and Lemma 3.5 states that this condition is sufficient. On the other hand, if there is more than one leader, or at least one processor finds an unacceptable neighborhood (and from Lemma 3.5, if x is not the input string then this must happen), then a halt is sent and eventually all the processors will end their computation.

The overall bit complexity of these phases is $O(n \log n)$. These completes the proof of Theorem 3.2. ■

3.5 Lower Bounds

In this section we prove lower bounds for the string recognition problem. In the case of Kolmogorov random string we need only the lower bound of $\Omega(n \log n)$ of [MW86] and the equivalent boolean function formulation of the string recognition problem stated in the introduction. This lower bound implies the optimality of the algorithm given above. For the case of $x = w^t v$ we have the following;

Theorem 3.3 *Let $x = w^{\lfloor \frac{n}{k} \rfloor} v$ be a string of period k such that $|x| = n$, $|w| = k$ and v is a proper prefix of w . The lower bound for recognizing any n -ary string on an n processors anonymous oriented ring is $\Omega(n \log n + \frac{n^2}{k})$ if $v = \emptyset$, $\Omega(n \log n)$ if $v \neq \emptyset$.*

PROOF Let x be a string as in the statement of the theorem. The $\Omega(n \log n)$ bound follows from [MW86]. Hence we only need to prove the $\Omega(\frac{n^2}{k})$ lower bound for the case $v \neq \emptyset$.

By Theorem 5.1 in [ASW88] we know that a lower bound on the bit complexity of computing a function can be derived as follows.

Given two input configurations y and z of length n , every algorithm that computes the function on input y requires at least

$$\sum_{i=0}^{\alpha} SI(y, i), \quad (3.1)$$

messages in the worst case.

Here α is a constant, such that there exist two processors p and q in the ring, for which p has the same α -neighborhood in y as q in z , but their output (at the end of the algorithm) is different. $SI(y, i)$ is the symmetry index function of the string y , i.e., the minimum number of elements in a equivalent i -neighborhood for processors in the configuration y .

Consider the string recognition problem with an input string $x = w^{\lfloor \frac{n}{k} \rfloor}$, $v = \emptyset$ and run the algorithm with input configurations $y = x$ and $z = w^{\lfloor \frac{n}{k} \rfloor - 1} w_0 \dots w_{k-1} \bar{w}_k$. There exist two processors p in y and q in z , that have the same $\lfloor \frac{n}{k} \rfloor - 1$ -neighborhood, but their output for the problem is different since $y = x$ and $z \neq x$. On input y there are at least $\frac{\lfloor \frac{n}{k} \rfloor}{2}$ processors who see the same $\lfloor \frac{n}{k} \rfloor - 1$ -neighborhood, we have then:

$$\sum_{i=0}^{\alpha} SI(y, i) = \sum_{i=0}^{\lfloor \frac{n}{k} \rfloor - 1} \frac{\lfloor \frac{n}{k} \rfloor}{2} = \lfloor \frac{n}{2} \rfloor \frac{\lfloor \frac{n}{k} \rfloor}{2}, \quad (3.2)$$

i.e., the lower bound is $\Omega(\frac{n^2}{k})$. ■

We finally show a family of strings whose lower bound is tight within a logarithmic factor. For every k we choose the string $x = (0^{k-1}1)^{\lfloor \frac{n}{k} \rfloor}$. The lower bound is $\Omega(nk + \frac{n^2}{k} + n \log n)$, since $\Omega(\frac{n^2}{k} + n \log n)$ is obtained from Theorem 3.3, and from the following intuition which adds an $\Omega(nk)$ factor: given as input string x and $x' = (0^{k-1}1)^{\lfloor \frac{n}{k} \rfloor - 1} 0^k$, every processor must trivially wait nk messages on x' before one processor can detect that the string $x' \neq x$, and then broadcast it.

Chapter 4

Conclusion

In this thesis we have investigated two problems. In Chapter 2 we presented the problem of routing in distributed systems. In particular we investigated some compact routing techniques among which Boolean Routing Schemes are of interest. All compact routing techniques minimize time and space complexity. In Section 2.4.4 we introduced a new routing function for unidirectional and bidirectional chordal rings with one chord. We have shown that this function has a space complexity of $O(n \log n)$ bits and time complexity $O(n)$ steps. We have also proved that for the problem of routing on chordal rings (with a chord of length k) the algorithm *Greedy*, introduced in Sections 2.4.2 and 2.4.3 is more powerful than ILSs with cyclic labeling.

In Chapter 3 we presented some results related to the computation of boolean functions on an anonymous, asynchronous ring of processors. We introduced and solved the string recognition problem. We have shown a $\Theta(n \log n)$ bit complexity bound for Kolmogorov random strings, which in turn implies this same bit complexity for almost all strings. For strings of the form $x = w^t v$, $t \geq 0$, $|w| = k$, we have shown an $O(nk + n \log n)$ bit upper bound when $v \neq \emptyset$ and an $O(nk + \frac{n^2}{k})$ bit upper bound when $v = \emptyset$. When $v = \emptyset$ we have also shown an $\Omega(n \log n + \frac{n^2}{k})$ bit lower bound which shows that the upper bound is tight, for $k = O(\sqrt{n})$. When $v \neq \emptyset$ the upper bound $O(nk + n \log n)$ is tight for $k = O(\log n)$.

4.1 Future Work

The problems we have introduced in this thesis offer new ideas for future work. An interesting open problem would be to find new topologies to which we could extend Boolean Routing Schemes. Given a particular topology and its properties we should be able to define functions with low worst case space/time complexity.

It would also be interesting to extend the function introduced in Chapter 3 to the case of chordal rings with more than one chord. Very little is known about routing on chordal rings with more than one chord. Some results related to IRSs are presented in [FGS94].

Regarding the string recognition problem of Chapter 3, we point out that the proof given for the case in which x is a Kolmogorov random string, can be modified and extended to other networks like tori, meshes, etc.

Another interesting open problem would be to tighten the bounds presented above for periodic strings. It would also be interesting to find new boolean functions on anonymous, asynchronous rings, that have $\Theta(n \log n)$ bit space complexity (i.e., are part of a small class of functions that require as many bits).

Appendix A

Index Search

The following section introduces the algorithm Index Search defined in Section 2.4.2. See Section 2.4.2 for further details.

ALGORITHM A.1 : Index Search

var

a, i, i', j, s, s', s'', s''', m, m', g, temp1, temp2, temp3, temp4, temp5: integer;

{ i, i' and j are possible indices, m and m' are distances we have moved,

s is the distance on the ring that we view as a straight line, s', s''

are temporary distances, s''' the distance of the optimal index closest

to 0, g is the number of times we do jumps of length $\lfloor \frac{n}{k} \rfloor$.

(For the case of $a < \frac{k}{2}$, temp1 is the value of the node

corresponding to the actual optimal i, temp2 is used to check while we

stop if the i at this level was optimal, temp3 is a value of a node,

temp4 is a temporary value of i, temp5 is a difference of indices.)}

t: boolean;

{ t is used to jump out of the cycles }

G: array [0, ..., 2][0, ..., a] of integers;

begin

{ initialization }

t := true;

m, m', s, temp1 := 0;

```

g, temp2 := 1;
.
.
.
for i := 0 to 2 do
  for j := 0 to a do G[i, j] := 0;
i, i', j := 0;
{ i' is the index of the optimal position closest to 0 }
s''' = a;
{  $a > \lfloor \frac{n}{k} \rfloor + 1$  }
if  $a < \frac{k}{2}$  then
  begin
    i' := a;
  while t do
    begin
      if  $m + (\lfloor \frac{k}{a} \rfloor - 1)(\lfloor \frac{n}{k} \rfloor + 1) < k$  then
        { at least  $\lfloor \frac{k}{a} \rfloor$  jumps are inside  $\{0, \dots, k\}$  }
        begin
          m := m + ( $\lfloor \frac{k}{a} \rfloor - 1$ )( $\lfloor \frac{n}{k} \rfloor + 1$ );
          j := j +  $\lfloor \frac{k}{a} \rfloor - 1$ ;
          s := s + ( $\lfloor \frac{k}{a} \rfloor - 1$ )a;
          s' := s - (g - 1)k;
          { temporary distance in  $\{0, \dots, k\}$  }
          if  $\lfloor \frac{gk-s}{a} \rfloor \geq 1$  then
            begin
              if  $m + (\lfloor \frac{gk-s}{a} \rfloor)(\lfloor \frac{n}{k} \rfloor + 1) < k$  then
                { there are still jumps at this level since we are
                  moving forward and all the nodes we get before jumping backwards
                  are optimally reachable}
            begin

```

```

     $m := m + (\lfloor \frac{2k-s}{a} \rfloor)(\lfloor \frac{n}{k} \rfloor + 1);$ 
     $j := j + \lfloor \frac{2k-s}{a} \rfloor;$ 
     $s := s + \lfloor \frac{2k-s}{a} \rfloor a;$ 
     $s' := s' + \lfloor \frac{2k-s}{a} \rfloor a;$ 
    end
    else
    { we will not go around anymore since we would
      move more then  $k - 1$  steps in total }
    begin
     $m := m + \lfloor \frac{k-m}{a} \rfloor(\lfloor \frac{n}{k} \rfloor + 1);$ 
     $j := j + \lfloor \frac{k-m}{a} \rfloor;$ 
     $s := s + \lfloor \frac{k-m}{a} \rfloor a;$ 
     $s' := s' + \lfloor \frac{k-m}{a} \rfloor a;$ 
    if temp2 = g then  $G[2, temp1] := j;$ 
    { this index is a possible candidate for the final i }
    t := false
    { we want to end the cycle }
    end
    end
    if t and  $m + \lfloor \frac{n}{k} \rfloor < k$  then
    { we are jumping one more time around }
    begin
     $m := m + \lfloor \frac{n}{k} \rfloor;$ 
     $s' := s' + a - k;$ 
     $s'' := \min \{m, s'\};$ 
    if  $s'' = m$  then  $s''' = \min \{s'', s'''\};$ 
    if  $s''' = m$  then
    begin
     $i' = j;$ 
     $m' = m;$ 

```

```

    end
    { i' is the index of the optimal jump closest to 0,
      m' is the distance we have moved }
     $s := s + a;$ 
     $j := j + 1;$ 
    { we check if j is a new candidate, i.e. is still optimal up to here }
    if  $s - (g - 1)k > m$  then
      begin
         $temp1 := s - (g - 1)k;$ 
         $G[0, temp1] := m;$ 
         $G[1, temp1] := j;$ 
         $temp2 := g;$ 
        { index of the new optimal round }
         $g := g + 1$ 
      end
    else
      { we don't go around anymore, but we have to check
        whether the index of the previous jump was a possible
        candidate }
      begin
        if  $t$  and  $temp2 = g$  then
          begin
             $G[2, temp1] := j;$ 
             $t := false;$ 
          end
        end
      end
    end
    else {  $m + (\lfloor \frac{k}{a} \rfloor - 1)(\lfloor \frac{n}{k} \rfloor + 1) \geq k$  }
    { we don't jump around anymore }
    begin

```

```

     $m := m + \lfloor \frac{k-m}{a} \rfloor (\lfloor \frac{n}{k} \rfloor + 1);$ 
     $j := j + \lfloor \frac{k-m}{a} \rfloor;$ 
     $s := s + \lfloor \frac{k-m}{a} \rfloor a;$ 
     $t := false;$ 
    if temp2 = g then  $G[2, temp1] := j$ 
      { other possible candidate }
    end
  end
end
{ we check the index j between all the possible
candidates memorized inside the array}
temp1, temp3, temp4, temp5 := 0;
for j := 1 to a do
  if  $\neg(G[0, j] = G[1, j] = G[2, j] = 0)$  then
    { we have a possible candidate}
    begin
      if  $G[0, j] > temp1 + j - temp3$  then
        begin
          { we insert the right values: this situation happens only
          if we reached a node with a smaller index with a smaller j. It
          might be convenient to move along the ring now. }
           $G[0, j] := temp1 + j - temp3;$ 
           $temp5 := G[2, j] - G[1, j];$ 
           $G[1, j] := temp4;$ 
           $G[2, j] := temp4 + temp5;$ 
        end
      else
        { we have a new optimal value }
        begin
           $temp1 := G[0, j];$ 

```

```

    temp3 := j;
    temp4 := G[1, j];
    if G[2, j] > i then i := G[2, j];
      { this is the temporary optimal i }
    end
    { we have now the optimal i }
  end
else { (G[0, j] = G[1, j] = G[2, j] = 0) }
begin
  G[0, j] = G[0, j - 1] + 1;
  G[1, j] = G[1, j - 1] + 1;
  G[2, j] = G[2, j - 1] + 1;
end
end
else { the case  $a > \frac{k}{2}$  is similar }
end

```

References

- [AM87] H. Attiya and Y. Mansour, "Language Complexity on the Synchronous Anonymous Ring", *Theoretical Computer Science*, **53** 169-185, (1987).
- [ASW88] H. Attiya, M. Snir and M. Warmuth, "Computing on an Anonymous Ring", *Journal of the ACM*, **35** (4), 845-875, (1988).
- [BLT90] E.M. Bakker, J. van Leeuwen, R.B. Tan, "Prefix Routing Schemes in Dynamic Networks", TechReport RUU-CS-90-10, Dept. of Computer Science, Utrecht University, Utrecht, (1990).
- [BLT91] E.M. Bakker, J. van Leeuwen, R.B. Tan, "Linear Interval Routing", *Algorithms Review*, **2**, 45-61, (1991).
- [BB89] P.W. Beame and H.L. Bodlaender, "Distributed Computing on Transitive Networks: The Torus", *6th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, Springer Verlag LNCS, **349**, 294-303, (1989).
- [BMW94] H.L. Bodlaender, S. Moran, M.K. Warmuth, "The Distributed Bit Complexity of the Ring: from the Anonymous to the Non-Anonymous Case", *Information and Computation*, **108** (1), 34,50, (1994).
- [CL92] R. Cuyvers, R. Lauwereins, "Hardware Fault-Tolerance: Possibilities and Limitations offered by Transputers", chapter 8 from "Transputers in Real-time Control", G. W. Irwin, P. J. Fleming, Eds., Research Studies Press Ltd., Taunton, Somerset, England, 202-238, (1992).

- [FMR94] P. Ferragina, A. Monti and A. Roncato, "Trade-off between Computation Power and Common Knowledge in Anonymous Rings", *Pre-Proceedings of Colloquium on Structural Information and Communication Complexity*, Ottawa, Canada, May 16-18 (1994).
- [FGS94] M. Flammini, G. Gambosi, S. Salomone, "Boolean Routing", *Proceedings of the 7th Int. Workshop on Distributed Algorithms (WDAG)*, Springer Verlag LNCS, 725, 219-233, (1994).
- [FGS94] M. Flammini, G. Gambosi, S. Salomone, "Interval Labeling Schemes For Chordal Rings", *Pre-Proceedings of Colloquium on Structural Information and Communication Complexity*, Ottawa, May 16-18, (1994).
- [FG94] P. Fraigniaud, C. Gavoille, "A Characterization of Networks Supporting Linear Interval Routing", *ACM Conference on Principles of Distributed Computing*, Los Angeles, California, 216-224, (1994).
- [FJ88] G.N. Frederikson, R. Janardan, "Designing Networks with Compact Routing Tables", *Algorithmica*, Springer-Verlag New York Inc, 3 171-190, (1988).
- [GS91] O. Goldreich, L. Shrara, "On the Complexity of Global Computation in the Presence of Link Failures: the Case of Ring Configuration", *Distributed Computing*, 5, (3), 121-131, (1991).
- [IR81] A. Itai, M. Rodeh, "Symmetry Breaking in Distributed Networks", *ACM Symposium on Theory of Computing*, 150-158, (1981).
- [KK90] E. Kranakis and D. Krizanc, "Computing Boolean Functions on Anonymous Hypercube Networks", Report CS-R9040, Centre for Mathematics and Computer Science, Amsterdam, (1990).
- [KK91] E. Kranakis and D. Krizanc, "Distributed Computing on Anonymous Hypercube Networks", *Proceedings of the 3rd IFFE Symposium on Parallel and Distributed Processing*, Dallas, Dec. 2-5, 722-729, (1991).

- [KK92] E. Kranakis and D. Krizanc, "Computing Boolean Functions on Cayley Networks", *Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing*, Arlington, Texas, Dec. 1-4, 222-229, (1992).
- [KKL94] E. Kranakis, D. Krizanc, F.L. Luccio, "String Recognition on Anonymous Ring", TechReport-256, Carleton University, School of Computer Science, Ottawa, (1994).
- [KKR93] E. Kranakis, D. Krizanc, S.S.Ravi, "On Multi-Label Linear Interval Routing Schemes", *Graph-Theoretic Concepts in Computer Science, Proceedings of the 19th Int. Workshop WG*, Springer Verlag LNCS, 790, 338-349, (1993).
- [LT87] Jan van Leeuwen, R.B. Tan, "Interval Routing", *The Computer Journal*, 30, 4, 298-307, (1987).
- [LV90] M. Li, P.M.B. Vitanyi, "Kolmogorov Complexity and its Applications", *Handbook of Theoretical Computer Science, Algorithms and Complexity*. The MIT Press, Cambridge, Massachusetts, (1990).
- [MW86] S. Moran and M. K. Warmuth, "Gap Theorems for Distributed Computation", *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*, 131-140, (1986).
- [SK85] N. Santoro, R. Khatib, "Modeling and implicit routing in networks", *The Computer Journal*, 28, 5-8, (1985).
- [S86] J. Seiferas, "A Simplified Lower Bound for Context-free Language Recognition," *Information and Control*, 69, 255-260, (1986).
- [T92] A.S.Tanenbaum, "Modern Operating Systems", Prentice Hall, New Jersey, (1992).
- [T94] G. Tel, "Introduction to Distributed Algorithms", Cambridge University Press, Cambridge, (1994).

- [W81] L.D. Wittie, "Communication structures for large networks of microcomputers", *IEEE Transactions on Computers*, C-30, 264, (1981).
- [ZL70] A.K. Zvonkin, L.A. Levin, "The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by means of the Theory of Algorithms", *Russian Math. Surveys*, 25, 83-124, (1977).

END

2 1-1 1-9 5

FIN