SAPIENZA
UNIVERSITÀ DI ROMA

SAPIENZA, UNIVERSITÀ DI ROMA

DOTTORATO DI RICERCA IN INFORMATICA

XXI CICLO – 2008

Algorithmic Problems in Network Communication

Emanuele Guido Fusco

SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza, Università di Roma

Dottorato di Ricerca in Informatica

XXI Ciclo - 2008

Emanuele Guido Fusco

Algorithmic Problems in Network Communication

Thesis Committee

Prof. Tiziana Calamoneri (Advisor)
Prof. Rossella Petreschi
Prof. Luigi Vincenzo Mancini

Reviewers

Prof. Pierre Fraigniaud
Prof. Leszek Gąsieniec

Author's address:

Emanuele Guido Fusco
Dipartimento di Informatica
Sapienza, Università di Roma
Via Salaria 113, 00198 Roma, Italy
e-mail: fusco@di.uniroma1.it
www: http://www.dsi.uniroma1.it/~fusco

Abstract

This thesis deals with algorithmic aspects of communication in networks. The main focus of the thesis is to find factors which can influence significantly the efficiency in performing a given communication task.

When we approach a problem, the first thing we have to do in order to study it, is to find a good model. A model is good if it allows to remove all factors that are irrelevant and expose the core of the problem we want to address, while remaining close to the reality. During the modeling step, however, there are cases in which more than one assumption is reasonable, and thus different models can be developed to study the same task. This makes it interesting to investigate how different assumptions can influence the bounds on the level of efficiency that can be achieved.

The efficiency itself is something that has to be well defined. Depending on the computational and communication model we assume, different measures of performance may be more appropriate to evaluate the quality of a given algorithm or distributed protocol. Depending on the scenario we consider, moreover, we may have the need to optimize our solutions with respect to different objective functions. A protocol that performs a given task quickly could have a bad behavior with respect to, e.g., power consumption or fault tolerance. While in some cases we can focus on a single parameter, in many situations we need to search for solutions that provide good trade-offs among different performance metrics.

Another factor worth to mention is the amount of available information. Distributed protocols that rely on complete knowledge of the network are impractical. Indeed, nodes involved in a communication task have to cooperate in order to complete it, often relying on local knowledge to achieve a global optimum. The relation between the information provided to nodes and the efficiency is one of the main topics in this thesis.

We analyzed the broadcast operation, one of the basic network communication primitives, in different models and considering different performance metrics. We also studied the spanning trees with many leaves problem, which is related to power consumption issues for the broadcast operation in radio network, and the lifetime problem for repeated broadcast operations. Another task we studied is acknowledged broadcasting, that is a communication task closely related to the broadcast operation. Other problems addressed in this thesis are a coloring problem arising from radio-frequency assignments, the autonomuos deployment of mobile sensors, and the assignment of proxies to simulate defective devices in a parallel computer.

Preface

The original contribution presented in this thesis has been obtained during a three years Ph.D. program in Computer Science at the University of Rome "Sapienza".

The results presented in Sections 2.2, 2.3, and 4.3 have been achieved in collaboration with Prof. Andrzej Pelc, during a five months visit to the Université du Québec en Outaouais. Most of the contributions presented in Section 2.2 have been published in the proceedings of *SPAA'08* [82], while an extended version, also containing the contributions of Subsection 2.2.8, has been accepted for publication in *Algorithmica* [83]. The results presented in Section 2.3 appeared in *DISC'08* [81]. The journal version of this work is currently submitted to *Distributed Computing*. The results presented in Section 4.3 have been accepted for publication in *Information Processing Letters* [80] and are still to appear.

Other results that are still to appear are those presented in Section 2.4. These results have been obtained in collaboration with my advisor, Prof. Tiziana Calamoneri, and Prof. Andrzej Pelc, during a visit of Prof. Pelc to our department and have been accepted to *OPODIS'08* [25].

Section 3.2, is based on a joint work with Prof. Tiziana Calamoneri, Prof. Andrea Clementi, and Prof. Riccardo Silvestri, presented at *OPODIS'07* [24] while Section 3.3 is based on collaboration with Prof. Angelo Monti, presented at *ISAAC'07* [79].

The content of Section 4.2 comes from a joint work with Prof. Tiziana Calamoneri, Prof. Richard B. Tan, and Prof. Paola Vocca. This work has been carried out during a visit of Prof. Tan to our department. It has been presented at *SIROCCO'06* [27] and accepted for publication, in an extended form, in *Mathematical Methods of Operations Research* [28].

Section 5.2 is based on collaboration with Dr. Novella Bartolini, Prof. Tiziana Calamoneri, Prof. Annalisa Massini, and Dr. Simone Silvestri. A preliminary version of this work appeared as a short paper at *DCOSS'08* [13]. A version containing an improved algorithm has been accepted to *IWSOS'08* [12], while the full version will appear in *Wireless Networks* [14].

Finally, Section 5.3 is the result of a joint work with Prof. Tiziana Calamoneri, Prof. Anil M. Shende and Prof. Sunil M. Shende, carried out during a visit of Prof. Anil M. Shende to our department. This work has been published in the conference proceedings of *SIROCCO'07* [26].

Other publications I contributed to during my research as a Ph.D. student, that are not included in this thesis, are briefly mentioned below.

Starting from the work presented in Section 4.2 I developed an interest for the concept of treewidth and for k-trees. In collaboration with Dr. Saverio Caminiti and Prof. Rossella Petreschi, we proposed a bijective code for k-trees with linear time coding and decoding algorithms. This contribution has been presented at *ESCAPE'07* [32] and it has been accepted for publication, in an extended form, in *Theory of Computing Systems* [33] (available electronically, to be printed). In a joint work with Dr. Saverio Caminiti, published electronically in the *Journal of Integer Sequences* [31], we provided a recursive formula to compute the number of labeled k-arch graphs (a superclass of k-trees) on n nodes, for any k.

In a joint work with Prof. Rossella Petreschi and Prof. Irene Finocchi we studied algebraic hypercube colorings. This work has been initiated during my master degree thesis and it has been presented in *ITNG'07* [71].

Acknowledgments

I would like to thank my advisor, Tiziana, and all my coauthors: Novella, Saverio, Andrea, Irene, Annalisa, Angelo, Rossella, Anil, Sunil, Riccardo, Simone, Richard, and Paola. A special thank goes to Andrzej: working with him turned a long, snowy, and chilly Canadian winter into a wonderful experience.

Notation

We use the symbol □ to denote the end of a proof. ■ denotes the end of an algorithm, when described at a high level without pseudo-code. Similarly, ♣ denotes the end of a subroutine. We use log to denote the logarithm with base 2.

Contents

1 Introduction **1**

1.1 Structure of the Thesis . . . 3

2 Information vs. Efficiency Trade-offs in Broadcasting **7**

2.1 Introduction . . . 8

2.2 Broadcasting with Advice in Trees . . . 12

2.2.1 The Model and the Problem . . . 12

2.2.2 Related Work . . . 13

2.2.3 Original Contribution . . . 14

2.2.4 Terminology and Preliminaries . . . 14

2.2.5 Lower Bounds . . . 15

2.2.6 Upper Bounds . . . 18

2.2.7 The Trade-off Curve . . . 21

2.2.8 Information Cost of Optimal Broadcasting . . . 21

2.2.9 Conclusion . . . 23

2.3 UDG Radio Networks with Missing and Inaccurate Information . . . 24

2.3.1 The Model and the Problem . . . 24

2.3.2 Original Contribution . . . 25

2.3.3 Terminology and Preliminaries . . . 26

2.3.4 Broadcasting in Sparse Networks . . . 27

2.3.5 Broadcasting in Dense Networks . . . 32

2.3.6 The Main Algorithms . . . 34

2.3.7 Lower Bound on Universal Broadcasting Time . . . 36

2.3.8 Conclusion . . . 39

2.4 Impact of Information on Asynchronous Radio Broadcasting . . . 40

2.4.1 The Model and the Problem . . . 40

2.4.2 Centralized vs. Ad Hoc Broadcasting . . . 40

2.4.3 Adaptive vs. Oblivious Protocols . . . 41

2.4.4 Original Contribution . . . 41

2.4.5 Terminology and Preliminaries . . . 42

2.4.6 UDG Radio Networks . . . 43

2.4.7 Symmetric Networks of Known Topology . . . 47

2.4.8 Networks of Unknown Topology . . . 48

2.4.9 Broadcasting Against the Weak Adversary . . . 49

2.4.10 Conclusion . . . 51

3 Power Consumption **53**
3.1 Introduction . . . 54
3.2 Random Geometric Radio Networks: Broadcast Lifetime . . . 56
3.2.1 The Model and the Problem . . . 56
3.2.2 Original Contribution . . . 57
3.2.3 Terminology and Preliminaries . . . 58
3.2.4 The Upper Bound . . . 59
3.2.5 The Algorithm . . . 59
3.2.6 The Distributed Protocol . . . 63
3.2.7 Conclusion . . . 66
3.3 Spanning Trees with many Leaves in Regular Bipartite Graphs . . . 67
3.3.1 The Model and the Problem . . . 67
3.3.2 Original Contribution . . . 67
3.3.3 Terminology and Preliminaries . . . 67
3.3.4 Regular Bipartite Graphs . . . 68
3.3.5 Cubic Bipartite Graphs . . . 70
3.3.6 NP Hardness . . . 72
3.3.7 Conclusion . . . 74

4 Other Communication Tasks **77**
4.1 Introduction . . . 78
4.2 $L(h,1,1)$-Labeling of Outerplanar Graphs . . . 81
4.2.1 The Model and the Problem . . . 81
4.2.2 Original Contribution . . . 81
4.2.3 Terminology and Preliminaries . . . 82
4.2.4 L(1,1,1)-Labeling . . . 84
4.2.5 L(h,1,1)-Labeling . . . 89
4.2.6 Conclusion . . . 91
4.3 Acknowledged Broadcasting in Ad Hoc Radio Networks . . . 92
4.3.1 The Model and the Problem . . . 92
4.3.2 Original Contribution . . . 92
4.3.3 AB in the Spontaneous Wake Up Model . . . 93
4.3.4 Impossibility of AB in the Conditional Wake Up Model . . . 96
4.3.5 Conclusion . . . 97

5 Fault Tolerance **99**
5.1 Introduction . . . 100
5.2 Autonomous Deployment of Mobile Sensors . . . 102
5.2.1 The Model and the Problem . . . 102
5.2.2 Related Work . . . 102
5.2.3 Original Contribution . . . 103
5.2.4 Algorithm `Push&Pull` . . . 103
5.2.5 A Discussion on Uniformity Implications . . . 108
5.2.6 Algorithm Properties . . . 111
5.2.7 Simulation Results . . . 115
5.2.8 Conclusion . . . 124
5.3 Proxy Assignments for Filling Gaps in Wireless Ad-hoc Lattice Computers . . . 125
5.3.1 The Model and the Problem . . . 125
5.3.2 Original Contribution . . . 125

5.3.3 Analogical Simulations 126
5.3.4 Terminology and Preliminaries 126
5.3.5 Single Row Gaps 128
5.3.6 Row-Column Convex Gaps 130
5.3.7 Conclusion 134

Chapter 1

Introduction

To establish how the efficiency in performing important communication tasks in networks can be influenced by different factors is the main goal of this thesis.

Our contributions are mainly based on algorithm design and analysis, using combinatorial and graph theoretical tools. To a minor extent, probabilistic tools and simulations have also been used. The approach used throughout the thesis to study network communication avoids to consider any particular device or networking protocol; indeed, all our contributions rely on simple network models, based on different graph families, in order to prove general results which remain valid regardless of the technology adopted.

In this introduction we would like to briefly describe some of the factors affecting efficiency in network communication that will be analyzed in the following chapters.

Different assumptions made during the **modeling step** can greatly affect the level of performance we can achieve in our solutions. As an example, depending on the model of communication we consider, the graphs used to model networks can be either directed or undirected. Undirected graphs can be used to describe the network topology whenever the reachability relation defined by the model of communication is symmetric. Radio networks, e.g., are studied both in symmetric and asymmetric models, as different characteristics of the transmitting devices can make one model more appropriate than the other. In wired networks, the reachability relation is more often assumed to be symmetric, as cables allow bidirectional communication. While protocols working correctly on directed graphs can be applied to undirected graphs, the opposite is not true. It follows that we can expect to have better performance when we work under a model of communication allowing us to restrict attention to undirected graphs.

Further restrictions on the type of graph modeling the network can be interesting to study. Some tasks can be NP hard in general, but they can admit an efficient solution in a restricted but still meaningful family of graphs. In some cases, the network topologies which are difficult to address may be very unlikely to arise in practice. A deterministic solution for a given problem has to be correct on all inputs, but if we allow a small probability of failure, using a probabilistic approach, then we can restrict attention to a subclass of instances having some good properties. In order to do so we need to be able to prove that these properties hold with high probability on instances which are relevant to the problem we are studying.

The definition of the model of communication is probably the most important part of the modeling step. There are problems however that are complex even to state; in such cases, a good model should allow to reason on a clean formulation of the problem itself. As an example, frequency assignment problems in radio networks become node coloring problems on graphs: the behavior of radio transmissions is handled by mean of coloring constraints, thus confining the complexity of the physics into the modeling step.

Once the problem we want to analyze is correctly modeled, we still need to define the **measure of performance** we consider in order to evaluate efficiency. The one that is used more frequently to evaluate centralized algorithm is the **asymptotic time complexity**. Such a measure provides a good estimation of the performance we can expect by an algorithm, when implemented and executed on a modern computer, only if the amount of data processed is small enough to fit in the random access memory. Complex memory hierarchies indeed require the algorithms to express good locality properties, as the number of accesses to the slow external memories, like hard disks (and in general memories on lower levels in the hierarchy), has a strong impact on the execution time.

When we consider distributed protocols, computation performed locally may be irrelevant with respect to the time spent in message exchanges, thus the communication complexity is often preferred to the time complexity in order to evaluate efficiency. Pushing this assumption to the limits however, as in models where NP complete problems can be solved locally in negligible time, makes the model impractical, and solutions achieved under such a set of assumptions only have a theoretical interest.

The choice of the measure of performance is thus strictly related with the model we assume. If the model fails to capture aspects that are important with respect to the task we are studying, then the measure of performance will fail to provide a good estimation of the actual efficiency of the algorithms developed on top of it.

In many situations, we cannot afford to consider performance of an algorithm with respect to a single parameter. Consider as an example approximating algorithms, usually developed in order to obtain "sufficiently good" solutions to NP hard problems, in polynomial time. Performance of such algorithms is evaluated with respect both to the time taken to output the solution and to the quality of the solution itself, in terms of distance from the optimum. This is still a simplification as, depending on the scenario, we may have the need to be able to choose the level of precision, trading completion time for the quality of the solution obtained. If this is the case, we can consider our problem solved and we can start to consider efficiency only if we have a polynomial-time approximation scheme.

Power consumption is an important measure of performance. Minimizing the power consumption of a network can indeed lower the total cost of ownership, resulting both in competitive advantages and in environmental friendliness. The minimization of overall power consumption can result in unbalanced workload among nodes. In the case of networks made of portable devices depending on batteries of limited capacity like, e.g., in a sensor network, it is important to maximize the **operating time** between battery recharges. To achieve this goal, early power drains in few nodes have to be avoided and thus the workload has to be balanced even at the cost of increasing overall power consumption.

The ability to **correct faults** (or to correctly operate in spite of their presence) becomes a fundamental goal when we deal with unreliable devices.

Obviously, the efficiency of a given algorithm or distributed protocol for a given problem can vary dramatically depending on the measure of complexity we adopt and the different parameters we consider for optimization.

Another factor affecting efficiency is the **availability of information**. Indeed, while a centralized algorithm operating over a network receives the network itself as part of its input, this is not the case for distributed protocols. In a distributed protocol, nodes have to operate usually relying on a partial and local knowledge of the network. This makes it interesting to investigate the relation between the availability of information and efficiency of the solution to a given problem. If we consider how much of the total knowledge is available to nodes in the network at the very beginning of the execution of the task, we have at the two extremes on the one hand the situation in which each node has complete knowledge of the network, and on the other hand the case where each node only knows its own label.

When complete knowledge of the network is available to all nodes, each node can simulate a centralized algorithm, and thus we are allowed to assume the existence of a central monitor even when we develop distributed protocols. The case where nodes only have a partial knowledge is more interesting, both because it models more faithfully real communication environments, and because it allows to define different submodels with clear separations in terms of achievable performance. Protocols can be either adaptive or oblivious. In the first case, nodes can acquire, during the execution of the protocol, additional knowledge of the network, and can modify their behavior depending on this information. The behavior of a node in an oblivious protocol is instead fully determined by the initial knowledge (available to the node since the very beginning of the execution of the task). It follows that we can expect adaptive protocols to outperform oblivious ones, while the opposite cannot happen.

Partial information available to nodes can be of many different kinds. It can be local knowledge as in the case where, e.g., nodes know their neighborhood within a certain distance, or it can be global, when nodes know global parameters of the network like the total number of nodes and the diameter. It is possible to study how the availability of a particular kind of information affects the efficiency, thus adopting what we can call a qualitative approach. If we change our perspective on the problem, we can see that the amount of available information (regardless of its type) can be a measure in itself. Indeed, we can use it in order to investigate the difficulty of a given task by analyzing how much information has to be provided to nodes in order to achieve a given degree of efficiency. In order to do so, we need to take a quantitative approach, abstracting from the particular knowledge provided to nodes.

1.1 Structure of the Thesis

This section contains a description of the structure of the thesis, together with a brief overview of the original contributions it contains. All the details concerning the problems addressed, and the related bibliographic references, are given in the respective chapters.

In Chapter 2 we focus on the relation between information and efficiency. The communication task we consider is the broadcast operation. This operation, which is one of the basic primitives in network communication, requires to disseminate a message from a distinguished node called the *source* to all nodes in the network.

Results presented in Section 2.2 are based on the *advice paradigm*, that is a recently introduced methodology enabling a quantitative approach for studying the relation between the amount of information available to nodes (or in general to agents operating in a distributed setting), and the efficiency in solving a given task. The task we analyzed is the broadcast operation in tree networks in the synchronous one-port model (one of the classical models for studying wired network communication). The measure of performance we considered is communication time. Our contribution is the first one to give the approximate shape of the whole curve representing the relation between available information and broadcasting time.

Section 2.3 is devoted to radio networks. In the model we used for such networks, broadcasting time depends on the diameter and on the density, i.e., the minimum Euclidean distance between two nodes. One more thing which influences performance is the ability of nodes to perceive their position with respect to an absolute coordinate system. This ability enables the usage of techniques, based on tilings of the Euclidean plane, which make it easier to achieve a good level of parallelism while avoiding collisions during communication. (We will see in the sequel how the problem of collisions affects broadcasting time in radio networks.)

While the assumption of having positional information is reasonable, since devices like GPS are nowadays cheap and small, it is practically fundamental and theoretically interesting to study

how the availability of **approximate** positional information can be exploited, as in the literature this information has always been assumed to be exact. Moreover, we dropped another assumption made in the literature, that the density of the network is known to nodes. It turned out that the combination of missing and inaccurate information (unknown density and unknown error in perceiving positions) substantially changes the problem with respect to the easier scenario previously embraced. The main challenge in this setting has been achieving fast broadcasting in sparse networks, where the optimal broadcasting time is linear with respect to the diameter.

Section 2.4 considers a different scenario and a different performance metric. While in Sections 2.2 and 2.3 we assumed a synchronous model of communication, in this section we assumed the communication to be asynchronous, adopting a recently introduced model for radio networks. Here the performance metric is the work spent to complete broadcasting, i.e., the total number of messages sent in the whole network (notice that the work coincide with the message complexity). Indeed, due to the fact that in this model we assumed that the adversary can delay arbitrarily the actual transmission of the messages with respect to the scheduled times, time is not a good measure of performance anymore. In this setting, we improved some of the existing results and we extended the study from centralized algorithms to distributed protocols, addressing the problem of asynchronous broadcasting in networks of unknown topology. Our analysis involves different graph families and considers both oblivious and adaptive protocols.

Chapter 3 addresses the issue of power consumption in network communication.

In Section 3.2 we consider the lifetime problem for repeated broadcast operations originating from a fixed source. As opposed to the rest of the thesis, the contribution described in this section relies on probabilistic tools. The assumption at the basis of this work is that actual radio networks like sensor networks can be faithfully modeled as random geometric graphs, where nodes are placed on a region in the Euclidean plane according to a uniform probability distribution and (oriented) edges between nodes are placed according to the transmitting range of the sender. Exploiting properties satisfied by such graphs **with high probability**, we have been able to develop a fully distributed protocol to perform a number of broadcast operations, originating from a fixed source, that is 1/12 of the optimum.

Section 3.3 considers the problem of finding a spanning tree with many leaves. If we consider a radio network, made of homogeneous nodes which cannot adjust their transmitting range, identifying a spanning tree of the topology graph with the maximum number of leaves allows us to perform broadcasting with the minimum overall power consumption. Indeed, information dissemination can be obtained by letting internal nodes in the spanning tree transmit the source message, while leaves can avoid to transmit. (Obviously, if the source is a leaf in the spanning tree, it still has to transmit in order to pass the message to at least one of the internal nodes, thus an optimal spanning tree with many leaves could provide a solution to the minimum energy broadcast problem which differs from the optimum by the cost of a single transmission.)

Unfortunately, the problem of finding a spanning tree with many leaves is NP hard. On the positive side we know that it can be approximated with a ratio of 2 in polynomial time. Our contribution to this topic consisted in studying an alternative formulation proposed in the literature, where the input graphs are d-regular (i.e., all nodes have degree d) and bipartite, and the goal is to find a spanning tree maximizing the number of leaves in one of the sets composing the partition. We proved that this variation of the problem is NP hard for any $d \geq 4$, and developed linear time approximating algorithms providing an approximation ratio of $2 - 2/(d-1)^2$ for d-regular bipartite graphs, for any $d \geq 3$.

Chapter 4 is devoted to two different communication tasks.

Section 4.2 considers a labeling problem related to radio networks. We already briefly mentioned that the main issue to solve in order to achieve fast communication in radio networks is the problem

of collisions. The methods we used, e.g., in Section 2.3, in order to achieve fast broadcasting are based on time shifts to avoid simultaneous transmissions of interfering nodes and require the network to be synchronous. As a result, communication protocols tend to be complex and task oriented (e.g., broadcasting protocols rely in some cases on the possibility of having a distinguished node, namely the source, in order to trigger some subtasks). A different approach to solve the problem of collisions is based on splitting the available radio spectrum into disjoint channels, in such a way that nodes assigned to different channels can transmit simultaneously and be heard by their recipients even though they are close to each other.

One of the main advantages of such a technique is that it is independent of the specific communication task. Communication protocols using radio-frequency assignment can neglect the problem of collisions. This makes it possible, e.g., to adopt protocols developed for wired networks, with point to point communication, having moreover the possibility to broadcast to all neighbors of a node in a single message.

The radio-frequency assignment problem can be modeled as a node coloring problem. However, differently from the usual definition of node coloring, which requires adjacent nodes to be assigned different colors, a typical channel assignment problem has more strict requirements.

A very general definition of node colorings arising from the radio-frequency assignment problem is the one of the $L(\delta_1, \ldots, \delta_\sigma)$-labelings. Colors are integers and the coloring constraints require to assign colors which are at least δ_i apart to nodes at distance i from each other, for $1 \leq i \leq \sigma$.

The drawback of splitting the radio spectrum in disjoint channels consists in the reduced bandwidth available for each channel, and consequently in a reduced speed of communication. It is thus fundamental to use as few channels (i.e., colors) as possible.

Our contribution to this topic consisted in a linear time algorithm for $L(h, 1, 1)$-labeling of outerplanar graphs. We present a lower bound on the number of colors needed to perform an $L(1, 1, 1)$-labeling of some outerplanar graphs, depending on the maximum node degree. Then we present a linear time coloring algorithm that finds an $L(1, 1, 1)$-labeling of any outerplanar graph using a number of color that is within a small additive constant from the lower bound provided. The algorithm is then extended to perform $L(h, 1, 1)$-labelings, for $h \geq 2$.

It should be noted that the $L(1, 1, 1)$-labeling problem is polynomial time solvable in outerplanar graphs, nevertheless the hidden constants in the execution time of the existing algorithm are too big to consider its implementation.

Section 4.3 addresses the *acknowledged broadcast* operation in radio networks. Acknowledged broadcasting is a communication task consisting in transmitting a message from the source to all other nodes of the network and making this fact common knowledge among all nodes. This task has been proved to be infeasible when nodes are unable to distinguish collisions in transmissions from silence (i.e., collision detection is not available), even when the network topology graph is symmetric. The infeasibility proof, however, was based on the assumption that the acknowledged broadcasting algorithms have to be correct on all networks, including the network composed only of the source. Uchida et al. showed an acknowledged broadcasting algorithm, without collision detection, working in time $O(n^{4/3} \log^{10/3} n)$ for all strongly connected networks of size at least 2 (for networks which are not strongly connected acknowledged broadcasting is infeasible). We improved the existing results for acknowledged broadcasting by showing algorithms that have the same time complexity as the best known broadcasting algorithms without acknowledgement, both with and without collision detection.

Reliability is an important factor to consider when we address problems related to network communication. In Chapter 5 we provide a brief introduction to fault tolerance. The contribution we present in Section 5.2 is a distributed protocol coordinating the self deployment of mobile sensors in a given area of interest. Such a deployment should guarantee complete sensing coverage of the

area and network connectivity among sensor. The deployment has a good behavior from the point of view of tolerance to faults, if both sensing and connectivity are maintained even in presence of a given number of defective sensors. We propose two variants of the protocol. In the first one, redundant sensors are used to perform self healing, adjusting their positions to fill sensing holes and to recover network connectivity after detection of a fault. In the second variant, redundant sensors are more uniformly spread and the resulting deployment maintains sensing coverage and connectivity even in presence of faults. In the latter case we provide a lower bound on the number of faults tolerated depending on the number of available sensors and the extension of the area of interest.

Section 5.3 considers proxy assignment schemes in 2-dimensional grid networks. Proxies are used to simulate defective devices and restore the network functionality, thus providing self healing capabilities. No redundancy is assumed, hence self healing is achieved at the cost of reduced performance in terms of computation and communication speed.

Chapter 2

Information vs. Efficiency Trade-offs in Broadcasting

2.1 Introduction

In this chapter we study the relation between information availability and efficiency of the *broadcast* operation in three different models.

The broadcast is a task initiated by a distinguished node called the *source* which has to disseminate a message to all nodes in the network. Remote nodes get the source message via intermediate nodes, along paths in the network. The importance of the broadcast operation is witnessed by the interest it gathered in the scientific community in the last more than twentyfive years.

The first two models we consider share the common assumption of having synchronous communication. This assumption is one of the most frequently used in the literature (see, e.g., [10, 11, 49, 50, 55, 56]). It requires all nodes to be equipped with internal clocks ticking at the same rate. Clocks measure consecutive time steps, called *rounds*. All clocks show the same round number at any given time. We assume that each node can send a message of arbitrary size in each round.

We are interested in broadcasting time. The performance metric we consider neglects the time spent in local computation. We analyze the communication time, in terms of the numbers of rounds needed to complete broadcasting. Depending on the further assumptions we make in the model of communication, we need to address different problems in order to achieve fast broadcasting.

In Section 2.2, we adopt the **synchronous one-port model** (also called the whispering model), introduced in [157]. In this model, every node can send an arbitrary message to at most one neighbor in each round. Moreover, we assume that only nodes that have already received the source message can transmit (this model is called *conditional wake up model* as opposite to the *spontaneous wake up model*, that allows nodes to transmit before being woken up). The latter assumption makes broadcasting equivalent to waking up the network, when in the beginning only the source is awake and other nodes are dormant, and has been also used for other models of communication (cf. e.g., [119, 145]).

We consider networks with a tree topology; the source message is thus available at the root of the tree and it is passed from parents to children along the tree. Under this set of assumptions, the speed of broadcasting is determined by the order chosen by nodes to inform their children. Indeed, if every node informs its children in decreasing order of optimal completion time of their respective subtree, broadcasting is completed in the optimal time.

The synchronous one-port model is a good way to model wired networks, where point to point transmissions are natural, but it is inadequate to model **radio networks**. In radio networks, addressed in Section 2.3, nodes do not have the ability to choose the recipients of their messages, as the communication media is shared and antennae are in general omnidirectional. A node can thus choose, in any round, either to act as a *transmitter* or as a *receiver*. A node acting as a transmitter in a given round sends a message delivered to all of its neighbors in the same round.

An important distinction from the receiver's point of view is between a message being just *delivered* and being *heard*, i.e., received successfully by a node. A node acting as a receiver in a given round *hears* a message if and only if a message from exactly one of its neighbors is delivered in this round. The message heard in this case is the one that was sent by the unique transmitting neighbor. If two neighbors v and v' of u transmit simultaneously in a given round, none of the delivered messages is heard by u in this round. In this case we say that a *collision* occurred at u. This model of communication was proposed in [41], where it was called *conflict-embodied*. In order to achieve fast broadcasting in radio networks, it is thus necessary to allow simultaneous transmissions of different nodes while minimizing collisions. This task can be simplified to a great extent when nodes have the availability of positional information [59], as, e.g., coordinates given by a GPS device.

The third model, considered in Section 2.4, drops the assumption of having synchronous com-

munication. A good reason to assume communication happens in an **asynchronous** way comes from the consideration that nodes in the network may be involved in more tasks than the lonely communication protocol, and thus they can insert unpredictable delays in transmissions, depending on the workload. Moreover, the assumption of having clocks agreeing on round number may be difficult to realize in practice (especially when we allow spontaneous transmissions of nodes that did not receive any message).

We model asynchrony by mean of an adversry that can insert an arbitrary delay between scheduled transmission time and actual transmission time. In such a setting, communication time is not a good measure of performance for broadcasting anymore. The model we used has been introduced in [46], together with a performance metric called work, i.e., the total number of transmissions made by nodes in the network in order to complete broadcasting.

The main focus of this chapter is on information, and how the availability of information can influence the efficiency of broadcasting.

The impact of available information about the network on the efficiency of communication algorithms is an important and well studied problem.

In [8] the authors showed a trade-off between the radius within which each node knows the topology of the network and the message complexity of broadcasting in the message passing model.

Algorithmic aspects of radio communication were mostly studied modeling a radio network as an arbitrary (directed or undirected) graph. These results can be partitioned into two subareas. In the case of radio networks, two parallel streams of research concerning communication algorithms have been recently very active: one assuming complete knowledge of the network (centralized communication) and the other assuming that each node knows only its own label (ad hoc networks).

Existing results concerning broadcasting time in radio networks show a significant difference between these two scenarios. The first paper to study deterministic centralized broadcasting in radio networks, assuming complete knowledge of the topology, was [41]. The authors also defined the graph model of radio network subsequently used in many other contributions. In [42], an $O(D \log^2 n)$-time broadcasting algorithm was proposed for all n-node networks of diameter D. This time complexity was then improved to $O(D+\log^5 n)$ in [84], to $O(D+\log^4 n)$ in [66], to $D+O(\log^3 n)$ in [92], and finally to $O(D + \log^2 n)$ in [121]. The latter complexity is optimal. On the other hand, in [4] the authors proved the existence of a family of n-node networks of constant diameter, for which any broadcasting algorithm requires time $\Omega(\log^2 n)$.

Investigation of deterministic distributed broadcasting in radio networks whose nodes have only local knowledge of the topology was initiated in [10]. The authors assumed that each node knows only its own label and labels of its neighbors. Several authors [22, 43, 44, 49, 50, 54, 56, 120] studied deterministic distributed broadcasting in radio networks under an even weaker assumption that nodes know only their own label (but not labels of their neighbors). In [43] the authors gave a broadcasting algorithm working in time $O(n)$ for all n-node networks, assuming that nodes can transmit spontaneously, before getting the source message. A matching lower bound $\Omega(n)$ on deterministic broadcasting time was proved in [120] even for the class of networks of constant radius.

In [43, 44, 49, 50, 56] the model of directed graphs was used. The aim of these papers was to construct broadcasting algorithms working as fast as possible in arbitrary (directed) radio networks without knowing their topology. The currently fastest deterministic broadcasting algorithms for such networks have running times $O(n \log^2 D)$ [56] and $O(n \log n \log\log n)$ [57]. On the other hand, in [54] an $\Omega(n \log D)$ lower bound on broadcasting time was proved for directed n-node networks of radius D.

Randomized broadcasting algorithms in radio networks were studied in [10, 56, 119, 125]. The authors do not assume that nodes know the topology of the network or that they have distinct

labels. In [10] the authors constructed a randomized broadcasting algorithm running in expected time $O(D \log n + \log^2 n)$. In [125] it was shown that for any randomized broadcasting algorithm and parameters $D \leq n$, there exists an n-node network of diameter D requiring expected time $\Omega(D \log(n/D))$ to execute this algorithm. The lower bound $\Omega(\log^2 n)$ from [4], for some networks of radius 2, holds for randomized algorithms as well. A randomized algorithm working in expected time $O(D \log(n/D) + \log^2 n)$, and thus matching the above lower bounds, was presented in [119] (cf. also [56]).

Another model of radio networks is based on geometry. Stations are represented as points in the plane and the graph modeling the network is no more arbitrary. It may be a unit disk graph, or one of its generalizations, where radii of disks representing areas that can be reached by the transmitter of a node may differ from node to node [59], or reachability areas may be of shapes different than a disk [62, 122]. Broadcasting in such geometric radio networks and some of their variations was considered, e.g., in [59, 62, 68, 89, 122, 150, 155]. The first paper to study deterministic broadcasting in arbitrary geometric radio networks with restricted knowledge of topology was [59]. The authors used several models, also assuming a positive knowledge radius, i.e., the knowledge available to a node, concerning other nodes inside some disk. In [68] the authors considered broadcasting in radio networks modeled by unit disk graphs, analyzing the differences between the spontaneous wake up model and the conditional wake up model.

Randomized broadcasting algorithms in geometric radio networks, where each node knows its position in the plane and the total number of nodes only, have been studied in [67]. In this work power consumption is also considered. The authors show that by assigning transmitting ranges to nodes in the network according to a power law distribution it is possible to achieve exponentially faster broadcasting, with respect to the model where all nodes have the same range, with an energy efficient algorithm.

If we consider a node receiving a message contemporaneously from more than one neighbor, we can chose one of the following two models. Either we can assume that the recipient of the messages, without being able to hear any of the messages sent, can distinguish the collision from silence, or we can assume it cannot. In the first case we say that collision detection is available, in the second case that it is not. Most of the above papers used the assumption that collision detection is not available. The paper [43] compared feasibility and efficiency of broadcasting with and without collision detection. The impact of collision detection on the efficiency of broadcasting for geometric radio networks was also discussed in [59].

The wakeup problem in radio networks was first studied in [91] for single-hop networks (modeled by complete graphs), and then in [45, 48] for arbitrary networks. In [110] the authors studied randomized wakeup algorithms for radio networks. In all these papers it was assumed that a subset of all nodes wake up spontaneously (possibly at different times) and have to wake up other (dormant) nodes. Waking up a radio network from a single source was studied in [145], in the case of anonymous networks. This is equivalent to broadcasting in the conditional wake up model.

Another network problem for which the impact of information on the efficiency of algorithms has been studied is network exploration, both in the anonymous [15] and in the labeled [58] setting. More generally, relations between knowledge concerning the environment and solution efficiency have been investigated in many areas of distributed computing: in [70, 132] a lot of impossibility results and lower bounds for distributed computing are surveyed, many of them depending on whether or not the nodes are provided with partial information concerning the topology of the network.

Asynchronous radio broadcasting was considered, e.g., in [46, 145]. In [46] the authors studied three asynchronous adversaries and investigated centralized oblivious broadcasting protocols working in their presence. They concentrated on finding broadcast protocols and verifying correctness of such protocols, as well as on providing lower bounds on their work. In [145] attention was focused on anonymous radio networks. In such networks not all nodes can be reached by a source message.

It was proved that no asynchronous algorithm unaware of network topology can broadcast to all reachable nodes in all networks.

2.2 Broadcasting with Advice in Trees

2.2.1 The Model and the Problem

In the papers cited in the introduction of this chapter, information about the network was of some particular kind, ranging from a numerical parameter, like (an upper bound on) the size of the network or its diameter, to the topology of the network within some radius from a node, and finally to the entire topology of the network which in fact makes distributed algorithms equivalent to centralized ones.

A different approach to the problem of evaluating the impact of information on the performance of network algorithms was adopted in [74], where the framework of network algorithms with *advice* has been introduced. The paradigm of network algorithms with advice can be described as follows. A priori, nodes of the network know only their own label and are able to distinguish ports leading to incident links. All other information about the network is coded as advice. An *oracle* knowing the entire network can give some strings of bits (advice) to some nodes (or to mobile agents in the case of exploration problems). Then a distributed algorithm is executed without knowing in which network it operates, using the provided advice. The total number of bits given by the oracle is the size of advice.

In this section we are concerned with broadcasting in trees, using advice.

A tree is a natural structure to perform broadcasting: even if the underlying network is more dense, some precomputed spanning tree of it, for example a minimum spanning tree, is often used to broadcast, for economy reasons. (A further discussion on this topic can be found in Chapter 3.)

The model of communication we assume is the synchronous one-port model. We recall that in this model, time is divided in rounds; nodes have individual clocks ticking at the same rate and showing the same round number. Every node can send an arbitrary message to at most one neighbor in a round. We also assume a conditional wake-up model, i.e., the source is the only node awake since round 1, while other nodes in the network become awake and can start to send messages after getting the source message.

All nodes have distinct labels $1, ..., n$, where n is the number of nodes. (Our results remain valid if distinct node labels are taken from a set $1, ..., N$, where $N \in O(n)$. Notice that some models in radio networks allow larger labels, i.e., from a polynomial range, see, e.g., [90]; the size of labels contributes only in a form of the logarithmic function.) We assume that, a priori, each node knows only its own label and can perceive ports $1, ..., d$ leading to its incident edges, where d is the degree of the node.

As opposed to papers on network algorithms with advice cited in Subection 2.2.2 below, we assume that the oracle can give advice not to all nodes but only to the root of the tree which is the source of broadcasting. In the case of the broadcasting task this is a natural assumption: it is easier to provide additional information about the network to one node only and, due to the nature of the broadcasting process and to the model assumptions, this information can be appended to the source message and be available at any node at the same time as the source message, hence as soon as the node is woken up and ready to use it.

Formally, an *oracle* is a function $\mathcal{O} : \mathcal{T} \longrightarrow \mathcal{F}$, where $\mathcal{T}$ is the set of all finite rooted trees and $\mathcal{F}$ is the set of finite binary sequences. The advice given by oracle $\mathcal{O}$ to the root of the tree T is the binary string $\mathcal{O}(T)$. The length of this string is the size of advice.

Since we want to evaluate the quality of a broadcasting algorithm using advice of given size and working for arbitrary trees, we have to define the measure of efficiency that we adopt. Intuitively, we seek algorithms that are fast in the worst case. However, adopting as a measure, say, the worst-case broadcasting time on the class of n-node trees does not seem appropriate. Broadcasting time is $n - 1$ on a n-node line or on a n-node star, regardless of the chosen algorithm, not because the algorithm is inefficient or the advice too small but because these trees are intrinsically long to inform

in the model of communication we assumed. Hence a more appropriate measure of efficiency of a broadcasting algorithm using advice but not knowing the tree should compare the time used by the algorithm to the optimal broadcast time that can be achieved using full knowledge of the tree.

This notion is similar to the competitive ratio used to evaluate on-line algorithms. In both cases, the performance of an algorithm lacking some essential knowledge about the environment is compared to that of an algorithm that has this knowledge: in the case of on-line algorithms, this knowledge concerns future events, and in the case of broadcasting in trees, it concerns the topology of the tree and its labeling.

Having such a definition of efficiency, the problem of measuring the impact of information on broadcasting time in trees can be formalized as finding the best competitive ratio of a broadcasting algorithm using advice of given size, for n-node trees.

2.2.2 Related Work

Broadcasting in the synchronous one-port model has been extensively studied in the literature. It is known that the problem of finding a minimum time broadcasting scheme for arbitrary graphs (with total information about the graph) is NP hard. Approximation algorithms for this problem were studied by numerous authors: in [117] an additive approximation with approximation summand $O(\sqrt{n})$ was given, and [65] gave the first approximation algorithm with sublogarithmic multiplicative approximation factor. In [148,157] it was shown that a minimum time broadcasting scheme for trees can be found in polynomial time (assuming full knowledge of the tree).

Research on network algorithm with advice has been quite active in the last few years. In [74] the framework of network algorithms with advice has been introduced and used to study the task of broadcasting with a linear number of messages, in the message passing model. Subsequently, this approach has been used to investigate various network problems: in [75] to study efficient exploration of networks by mobile agents, in [73] to study distributed graph coloring, in [76] to study the distributed minimum spanning tree construction, in [139] to study graph searching, and in [108] to study optimal time of broadcasting in radio networks.

It should be noted that a similar approach has been also used previously in the context of informative labeling schemes, cf. [1, 87, 112, 116, 160].

In [73–76,108,139] the authors studied the minimum size of advice required to solve the respective network problem in an efficient way. Thus the framework of advice permits to quantify the amount of information needed for an efficient solution of a given network problem, regardless of the type of information that is provided.

However, in all papers concerning the advice paradigm, the minimum size of advice was established only at some fixed level of efficiency of an algorithm solving the given problem.

In [74] the authors compared the minimum size of advice required to solve two information dissemination problems using a linear number of messages. In [75] the authors established the size of advice needed to break competitive ratio 2 of an exploration algorithm in trees. In [76] it was shown that advice of constant size permits to carry on the distributed construction of a minimum spanning tree in logarithmic time. In [73] the authors established lower bounds on the size of advice needed to beat time $\Theta(\log^* n)$ for 3-coloring of a cycle and to achieve time $\Theta(\log^* n)$ for 3-coloring of unoriented trees. It was also shown that, both for trees and for cycles, advice of size $\Omega(n)$ is needed to 3-color in constant time. In the case of [139] the issue was not efficiency but feasibility: it was shown that $\Theta(n \log n)$ is the minimum size of advice required to perform monotone connected graph clearing. Finally, in [108] the authors analyzed radio networks for which it is possible to perform broadcasting in constant time, proving that $O(n)$ bits of advice allow to obtain optimal time in such networks, while $o(n)$ bits do not suffice.

Hence, if we consider the curve representing the dependence of the cost of solving a network

problem (measured by time, by the number of messages, by the competitive ratio, etc.) on the size of advice, each of the above papers plotted only one or two points on the respective curve. Since such a curve represents the trade-off between available information and the efficiency of solving a given network problem, it is natural to ask for the entire curve, or at least for its approximate shape: the shape of the curve shows how sensitive to information the given problem is.

The concept of information sensitivity was introduced in [73]: a problem is information sensitive if few bits of advice suffice to drastically improve the efficiency of solving it. Since the authors of [73] proved a very large lower bound on the size of advice needed to improve time of 3-coloring in cycles, as compared to the time without advice, they concluded that this problem is information insensitive. Such a conclusion from only two points plotted in the curve is rare: it is justified only if, as in the case of 3-coloring of cycles, a large left segment of the curve is flat. In general, in order to learn how sensitive to information is a given network problem, we have to establish an approximate shape of the whole curve by giving close upper and lower bounds on the solution cost for all sizes of advice.

2.2.3 Original Contribution

Our goal is to find a trade-off between the size of advice given by the oracle to the source of broadcasting and the best competitive ratio of a broadcasting algorithm for n-node trees. We establish such a trade-off with an approximation factor of $O(n^\epsilon)$, for an arbitrarily small positive constant ϵ. It turns out that for q bits of advice the best competitive ratio is roughly $\Theta(\sqrt{n})$ for q up to $\sqrt{n}$, and it is roughly $\Theta(n/q)$ beyond this threshold. More precisely, we show that for any advice of size $q \leq \sqrt{n}$, the competitive ratio of any broadcasting algorithm using this advice is $\Omega(n^\gamma)$, for any $\gamma < 1/2$. On the other hand, any broadcasting algorithm (even without advice) has competitive ratio $O(\sqrt{n})$.

For larger advice, when $q > \sqrt{n}$, we get a lower bound $\Omega(n^{1-\epsilon}/q)$ on competitive ratio, for an arbitrarily small positive constant ϵ, and we show an oracle giving advice of size q and an algorithm using this advice with competitive ratio $O((n \log^2 n)/q)$.

Hence, for any size q of advice, the gap between our upper and lower bounds on the corresponding best possible competitive ratio is $O(n^\epsilon)$. In terms of information sensitivity of broadcasting time in trees our results show that this problem is information insensitive for low size of advice (up to $\Theta(\sqrt{n})$) and becomes information sensitive beyond this threshold.

We also study the minimum size of advice required to broadcast in optimal time (i.e., to achieve competitive ratio 1). For general n-node trees, we show an upper bound $O(n \log n)$ and a lower bound $\Omega(n)$ on this minimum size of advice. Moreover, we prove a tight bound $O(n)$ for two subclasses of trees.

2.2.4 Terminology and Preliminaries

Let T be a tree rooted at node r which is the source of broadcasting. Notions of *children* and *parent* of a node are used with respect to this rooted tree. The down-degree of an internal node v of T is the number of its children and it is denoted by $\delta(v)$. For any broadcasting algorithm A and any internal node v of T, we denote by $A(v)$ the time taken by algorithm A to complete the broadcast on the subtree of T rooted at v. The time taken by the algorithm to complete the broadcast in the whole tree is $A(T) = A(r)$. We denote by $\mathrm{opt}(v)$ the optimal time to complete the broadcast on the subtree rooted at v, with the full knowledge of its topology. The optimal broadcast time for the entire tree T is $\mathrm{opt}(T) = \mathrm{opt}(r)$.

As we previousy stated, our notion of efficiency for a given algorithm A is based on the competitive ratio between the time taken by A to complete the broadcast and the optimal broadcasting time. In particular, we seek for algorithms having good performance in the worst case, thus we

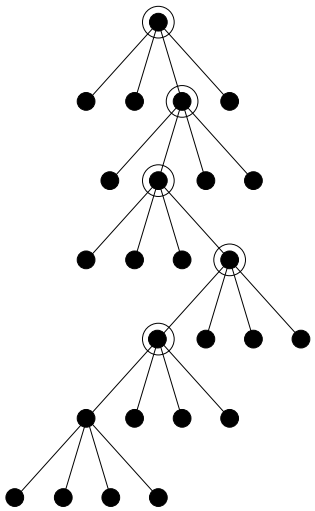

Figure 2.1: An instance of a $(5, 4)$-tree. Special nodes are surrounded by a circle.

consider, for a given algorithm A and a given number n, the worst case ratio of the time taken by A on a tree T over the optimal broadcasting time in T, where T is an arbitrary n-node tree. In the following we formally define our notion of competitive ratio, which we denote as $\mathrm{cr}\,(A, n)$.

Definition 2.2.1 *For a given broadcasting algorithm A and a fixed n, $cr(A, n)$ is the the maximum of $A(T)/\operatorname{opt}(T)$ over all n-node trees T.*

The *surplus* of an algorithm A on a tree T is $A(T) - \operatorname{opt}(T)$. Denote by $N(v)$ the number of nodes in the subtree of T rooted at v; the number of nodes in the whole tree T is $N(T) = N(r)$.

2.2.5 Lower Bounds

In this subsection we prove lower bounds on the competitive ratio of a broadcasting algorithm using advice of a given size. For this purpose we will use the following class of trees. A tree T with $n = (k+1)d+1$ nodes is called a (k, d)-*tree*, if and only if, it is of height $k+1$ and has $k+1$ internal nodes of down-degree d. We call *special* the internal nodes of T having a child that is not a leaf.

Notice that in a (k, d)-tree every internal node has exactly d children and at most one of them is an internal node. There are exactly k special nodes in a (k, d)-tree.

An instance of a (k, d)-tree is an assignment of a port $p \in \{1, \ldots, d\}$ leading to the non-leaf child at all special nodes (see Fig. 2.1, where ports are numbered left to right). It follows that there are d^k instances of a (k, d)-tree.

Lemma 2.2.1 *For any broadcasting algorithm A using $o(k \log d)$ bits of advice, for any $\alpha < 1$ and for sufficiently large n, there exists a n-node (k, d)-tree T such that $A(T)$ is at least:*

$$\begin{aligned} kd^{\alpha}, &\qquad \text{if } k \leq d \\ dk^{\alpha}, &\qquad \text{if } k \geq d \end{aligned}$$

Proof: We will use the following combinatorial fact.

Claim: The number of representations of an integer $s > 0$ as an ordered sum of r positive integers is $\binom{s-1}{r-1}$.

Proof of the claim: Every such representation $S = a_1 + \ldots + a_r$ can be coded as the following binary sequence with s zeros and $r-1$ ones inserted between them:

$$\underbrace{00\ldots0}_{a_1}\ 1\ \underbrace{00\ldots0}_{a_2}\ 1\ \ \ldots\ \ 1\ \underbrace{00\ldots0}_{a_r}$$

which proves the claim (e.g. $10 = 5+1+3+1$ can be coded as 00000 1 0 1 000 1 0).

Let $T(s,k)$ be the number of ways to put s identical balls into bins labeled $\{1,\ldots,k\}$. In view of the claim we have, for $s \geq 1$,

$$T(s,k) = \sum_{r=1}^{\min(s,k)} \binom{s-1}{r-1}\binom{k}{r} \tag{2.1}$$

because the number r of non empty bins can range from 1 to $\min(s,k)$, there are $\binom{k}{r}$ ways to choose the non empty bins and, for any such choice, the number of ways to put the s balls is equal to the number of representations of s as an ordered sum of r summands. We also have $T(0,k) = 1$.

Suppose that we have q bits of advice at the root of a n-node (k,d)-tree. (Recall that $n = (k+1)d+1$.) Hence all instances are colored with 2^q colors. Fix an algorithm A using this advice. For a fixed instance I and a fixed color there is a fixed *surplus pattern* $\sigma(I)$ associated with I, where the surplus pattern is defined as follows: $\sigma(I) = (a_1,\ldots,a_k)$, where $a_i \geq 0$ is the number of leaves that the i-th special node (counting from the root) has informed before informing its only internal child. In other words, a_i is the time spent by the i-th special node before informing its child corresponding to the largest subtree. It is easy to see that if $I \neq I'$ are given the same color, then $\sigma(I) \neq \sigma(I')$. The *total surplus* corresponding to the surplus pattern $(a_1,\ldots,a_k)$ is defined as $s = a_1 + \ldots + a_k$.

For a given color and a total surplus s, we say that an instance I causes this surplus if $a_1 + \ldots + a_k = s$, where $\sigma(I) = (a_1,\ldots,a_k)$. Hence, for a given color, the number of instances causing the surplus s is at most $T(s,k)$.

Let $\tilde{T}(\mathcal{S},k)$ be the number of instances causing surplus at most $\mathcal{S}$. We have

$$\tilde{T}(\mathcal{S},k) \leq \sum_{s=0}^{\mathcal{S}} T(s,k)$$

and

$$T(s,k) = \sum_{r=1}^{\min(s,k)} \binom{s-1}{r-1}\binom{k}{r} \leq \sum_{r=1}^{k} \binom{s}{r}\binom{k}{r} \leq \sum_{r=1}^{k} \binom{s}{r}\binom{k}{k/2} = \binom{k}{k/2}\sum_{r=1}^{k}\binom{s}{r}.$$

(For k odd, we denote by $\binom{k}{k/2}$ the number $\binom{k}{\lfloor k/2\rfloor} = \binom{k}{\lceil k/2\rceil}$.)

For $s \geq 2k$ we have

$$\binom{k}{k/2}\sum_{r=1}^{k}\binom{s}{r} < \binom{k}{k/2}k\binom{s}{k}$$

and for $s \leq 2k$ we have $\binom{s}{r} \leq \binom{s}{s/2} \leq \binom{2k}{k}$.

Since, for $S < S'$, guaranteeing surplus at most S is not easier than guaranteeing surplus at most S', it is enough to restrict attention to the case $\mathcal{S} \geq 2k$. Under this assumption we have $\binom{2k}{k} \leq \binom{\mathcal{S}}{k}$ and thus we obtain

$$\tilde{T}(\mathcal{S},k) \leq 1 + \sum_{s=1}^{\mathcal{S}} T(s,k) = 1 + \sum_{s=1}^{2k} T(s,k) + \sum_{s=2k+1}^{\mathcal{S}} T(s,k) \leq$$

$$\leq 2k \cdot k\binom{k}{k/2}\binom{2k}{k} + (\mathcal{S}-2k)\,k\binom{k}{k/2}\binom{\mathcal{S}}{k} \leq$$

$$\leq \mathcal{S}k\binom{k}{k/2}\binom{\mathcal{S}}{k} \leq \mathcal{S}k\left(\frac{2ke}{k}\right)^{k/2}\left(\frac{\mathcal{S}e}{k}\right)^{k} = \mathcal{S}k\left(\frac{\mathcal{S}}{k}e\sqrt{2e}\right)^{k},$$

where we used the inequality $\binom{x}{y} \leq \left(\frac{xe}{y}\right)^{y}$.

In order to show that with $q \in o(k \log d)$ bits of advice it is impossible to guarantee surplus at most $\mathcal{S}$, it is enough to prove (for sufficiently large n):

$$2^{q}\mathcal{S}k\left(\frac{\mathcal{S}}{k}e\sqrt{2e}\right)^{k} < d^{k} \tag{2.2}$$

because this will show that, using the provided advice, the number of instances causing surplus at most $\mathcal{S}$ is smaller than the total number of instances, and hence, for any algorithm using this advice, the adversary can choose one of the exceeding instances, thus causing the algorithm to exceed the surplus. Let $C = e\sqrt{2e}$ and consider two cases:

- **case 1:** $k \leq d$. If k is constant, then the lemma is true, so suppose $k \to \infty$. Let $\mathcal{S} \leq kd^{\alpha}$, for any $\alpha < 1$. For sufficiently large n we have

$$q + \log(\mathcal{S}k) + k\log\left(\frac{\mathcal{S}}{k}\right) + k\log C \leq$$

$$\leq q + \log\left(k^{2}d^{\alpha}\right) + k\alpha\log d + k\log C < k\log d$$

because $q \in o(k \log d)$ and $\alpha < 1$. This proves (2.2) in this case.

- **case 2:** $d < k$. If d is constant, then the lemma is true, so suppose $d \to \infty$. Let $\mathcal{S} \leq dk^{\alpha}$, for any $\alpha < 1$. For sufficiently large n we have

$$q + \log(\mathcal{S}k) + k\log\left(\frac{\mathcal{S}}{k}\right) + k\log C \leq$$

$$\leq q + \log\left(dk^{\alpha+1}\right) + k\log\left(\frac{d}{k^{1-\alpha}}\right) + k\log C < k\log d$$

because $q \in o(k \log d)$ and

$$k\log\left(\frac{d}{k^{1-\alpha}}\right) < k\log\left(\frac{d}{d^{1-\alpha}}\right) = \alpha k\log d.$$

This proves (2.2) in this case.

□

Corollary 2.2.1 *Let k be in $\Omega(\sqrt{n})$ and in $O(n^{\gamma})$, where γ is any constant such that $1/2 \leq \gamma < 1$. For any broadcasting algorithm A using $o(k \log n)$ bits of advice, there exists a tree T, such that $A(T) \in \Omega\left(n^{\beta}\right)$ for any $\beta < 1$ and $\mathrm{opt}(T) \in O(k)$. Hence $cr(A,n) \in \Omega\left(\frac{n^{1-\epsilon}}{k}\right)$, for any $\epsilon > 0$.*

Proof: Recall that for a n-node (k,d)-tree we have $kd \in \Theta(n)$. Thus, under the assumption $k \in O(n^{\gamma})$, it follows from Lemma 2.2.1 that, for any broadcasting algorithm A using $o(k \log n)$ bits of advice, for any $\alpha < 1$ and for sufficiently large n, there exists a n-node (k,d)-tree T such that $A(T) \in \Omega\left(dk^{\alpha}\right)$ (if $k \geq d$) or $\Omega\left(d^{\alpha}k\right)$ (if $k \leq d$). When $k \in \Omega(\sqrt{n})$, the optimal broadcasting time in any n-node (k,d)-tree is in $\Theta(k)$. Finally, $\alpha < 1$ implies that $(kd)^{\alpha} \leq k^{\alpha}d$ and $(kd)^{\alpha} \leq kd^{\alpha}$. It follows that $A(T) \in \Omega\left(n^{\beta}\right)$, thus proving the corollary. □

Corollary 2.2.2 *No broadcasting algorithm A using $o(n)$ bits of advice can achieve constant competitive ratio.*

Proof: Suppose that $q \in o(n)$ and that there exists a broadcasting algorithm A with competitive ratio $\beta \in \Theta(1)$ using q bits of advice. Let d be a sufficiently large constant and consider a (k, d)-tree T. Since $\mathrm{opt}(T) \leq k + d \leq 2k$, in order to guarantee constant competitive ratio we would need to guarantee a surplus $\mathcal{S} \leq 2\beta k$. Similarly as above this is impossible because

$$q + \log(\mathcal{S}k) + k \log\left(\frac{\mathcal{S}}{k}\right) + k \log C \leq$$

$$\leq q + \log(2\beta k^2) + k \log(2\beta C) < k \log d,$$

for sufficiently large d. (Notice that, if d is constant, k is in $\Theta(n)$, so q must be in $o(k \log d)$.) □

2.2.6 Upper Bounds

In this subsection we establish upper bounds on the best competitive ratio of a broadcasting algorithm using advice of given size. We do this by constructing an oracle which, for every tree, provides advice of given size, and an algorithm using this advice whose competitive ratio serves as the upper bound. For any size of advice, our upper bounds exceed the lower bounds from Section 2 only by a factor $O(n^\epsilon)$, for an arbitrarily small positive constant ϵ.

We start with the observation that $\sqrt{n}$ is an upper bound on the competitive ratio of any broadcasting algorithm for n-node trees. This upper bound is within the promised approximation ratio for advice of size up to $\sqrt{n}$.

Lemma 2.2.2 *The time of any broadcasting algorithm A (even using no advice) is at most h times larger than optimal on any tree of height h.*

Proof: Let T be a tree of height h, and let v be the last leaf informed by algorithm A. Let k be the length of the branch of v, and let $d_1, \ldots, d_k$ be the down-degrees of the internal nodes in this branch. Let $d_M = \max\{d_1, \ldots, d_k\}$. We have $A(T) \leq D = d_1 + \ldots + d_k \leq k d_M$ and $\mathrm{opt}(T) \geq d_M \geq D/k$. Hence

$$\frac{A(T)}{\mathrm{opt}(T)} \leq \frac{D}{D/k} = k \leq h.$$

□

Lemma 2.2.3 *The competitive ratio of any broadcasting algorithm A for n-node trees (even using no advice) is at most $\sqrt{n}$.*

Proof: Let T be any n-node tree and let h be its height. If $h \leq \sqrt{n}$ then $\frac{A(T)}{\mathrm{opt}(T)} \leq h \leq \sqrt{n}$ by Lemma 2.2.2. If $h > \sqrt{n}$ then $\mathrm{opt}(T) > \sqrt{n}$. On the other hand $A(T) \leq n$ thus proving the lemma. □

In order to get good upper bounds on competitive ratio of algorithms when the size of advice is large, we will use *d-wise algorithms*, defined as follows. A broadcasting algorithm A is *d-wise* if, for every tree T and any node v of down-degree greater than d, node v informs first its child w for which $A(w) = \max\{A(w') : w' \text{ child of } v\}$. Hence a d-wise algorithm disregards the order of informing children of nodes of down-degree at most d, and for nodes of larger down-degrees it first informs the child that is the root of the "most expensive" subtree while disregarding the order of informing other children of such nodes. Intuitively, disregarding the order at nodes of small down-degree is a

"wise" decision, as not much harm can be done even when the order of informing children of such nodes is completely wrong. Informing first the "most expensive" child while disregarding the order of other children at nodes of large down-degree limits the damage done at such nodes, even if the order of informing other children is not optimal. It turns out that these two decisions permit to save a lot of advice bits while keeping the competitive ratio under control.

Lemma 2.2.4 *The competitive ratio of a d-wise algorithm for n-node trees is at most* $d \log n$.

Proof: We prove the inequality $A(T)/\operatorname{opt}(T) \leq d \log n$ by induction on the height of the tree. For any tree of height 1 the ratio on the left-hand side of the inequality is 1. As any tree of height 1 has at least 2 nodes, we have $\log n \geq \log 2 = 1$.

By the inductive hypothesis, for any tree T' with n' nodes and height $\leq h$, we can assume $d \log n' \geq A(T')/\operatorname{opt}(T')$.

For any node v, let v_M be a child of v that maximizes $A(w')$, over all children w' of v. Moreover for any v of down-degree at least 2, let v_m be the child of v that maximizes $A(w')$, over all children $w' \neq v_M$ of v.

Let T be a n-node tree of height $h+1$, with root r. We consider several cases:

Case 1. $\delta(r) > d$.

In this case $A(T) \leq \max\{A(v_M)+1, A(v_m)+\delta(r)\}$.
If $A(v_M)+1 \geq A(v_m)+\delta(r)$ then

$$\frac{A(T)}{\operatorname{opt}(T)} \leq \frac{A(v_M)+1}{\operatorname{opt}(v_M)+1} \leq \frac{A(v_M)}{\operatorname{opt}(v_M)} \leq d \log N(v_M) < d \log n,$$

as $A(v_M) \geq \operatorname{opt}(v_M)$ and $(a+1)/(b+1) \leq a/b$ when $a \geq b$.
If $A(v_m)+\delta(r) > A(v_M)+1$ then:

- if $\frac{A(v_M)}{\operatorname{opt}(v_M)} \geq \frac{A(v_m)}{\operatorname{opt}(v_m)}$ then

  $$n > 2^{\frac{A(v_M)}{d \operatorname{opt}(v_M)}} + 2^{\frac{A(v_m)}{d \operatorname{opt}(v_m)}} \geq 2^{\frac{A(v_m)}{d \operatorname{opt}(v_m)}+1}.$$

  We have

  $$\frac{A(T)}{\operatorname{opt}(T)} \leq \frac{A(v_m)+\delta(r)}{\operatorname{opt}(T)} = \frac{A(v_m)}{\operatorname{opt}(T)} + \frac{\delta(r)}{\operatorname{opt}(T)} < \frac{A(v_m)}{\operatorname{opt}(v_m)} + 1 \leq \frac{A(v_m)}{\operatorname{opt}(v_m)} + d < d \log n,$$

  as $\delta(r) \leq \operatorname{opt}(T)$ and $\operatorname{opt}(v_m) < \operatorname{opt}(T)$;

- if $\frac{A(v_m)}{\operatorname{opt}(v_m)} > \frac{A(v_M)}{\operatorname{opt}(v_M)}$ then

  $$n > 2^{\frac{A(v_M)}{d \operatorname{opt}(v_M)}} + 2^{\frac{A(v_m)}{d \operatorname{opt}(v_m)}} \geq 2^{\frac{A(v_M)}{d \operatorname{opt}(v_M)}+1}.$$

  We have

  $$\frac{A(T)}{\operatorname{opt}(T)} \leq \frac{A(v_m)+\delta(r)}{\operatorname{opt}(T)} = \frac{A(v_m)}{\operatorname{opt}(T)} + \frac{\delta(r)}{\operatorname{opt}(T)} < \frac{A(v_m)}{\operatorname{opt}(v_M)} + 1 \leq \frac{A(v_M)}{\operatorname{opt}(v_M)} + d < d \log n,$$

  as $A(v_M) \geq A(v_m)$, $\delta(r) \leq \operatorname{opt}(T)$ and $\operatorname{opt}(v_M) < \operatorname{opt}(T)$.

Case 2. $\delta(r) \leq d$.

If $\delta(r) = 1$ then

$$\frac{A(T)}{\text{opt}(T)} = \frac{A(v_1)+1}{\text{opt}(v_1)+1} \leq \frac{A(v_1)}{\text{opt}(v_1)} \leq d \log N(v_1) < d \log n.$$

If $\delta(r) \geq 2$ then we have $A(T) \leq A(v_M) + d$.

- If $A(v_M) \geq d\,\text{opt}(v_M)$ then

$$\frac{A(T)}{d\,\text{opt}(T)} \leq \frac{A(v_M)+d}{d\,\text{opt}(T)} \leq \frac{A(v_M)+d}{d\,\text{opt}(v_M)+d} \leq \frac{A(v_M)}{d\,\text{opt}(v_M)} \leq \log N(v_M) < \log n.$$

- If $A(v_M) < d\,\text{opt}(v_M)$ then $A(v_M)/\,\text{opt}(v_M) < d$ and we have

$$\frac{A(T)}{\text{opt}(T)} \leq \frac{A(v_M)+d}{\text{opt}(T)} \leq \frac{A(v_M)}{\text{opt}(v_M)} + 1 \leq d+1 \leq d \log n,$$

as $n \geq 4$. □

Lemma 2.2.5 *There exists a d-wise algorithm A using $O\left(\frac{n \log n}{d}\right)$ bits of advice for n-node trees.*

Proof: Consider a n-node tree T and a sequence $(\ell_1, p_1), (\ell_2, p_2), \ldots, (\ell_m, p_m)$, where $\ell_i : i \leq r$ are the labels of nodes of T of down-degree $> d$, and p_i are port numbers. Let algorithm A be such that, at any node v with label ℓ_i, it informs first the child corresponding to port p_i, for $i \in \{1, \ldots, r\}$, and uses the order of increasing port numbers to inform the remaining children of v and to inform all children of nodes of down-degree $\leq d$.

The advice given to the root is a binary string of the form:

$$S = [\ \underbrace{11\ldots1}_{\lceil \log n \rceil}\ 0] \frown \ell_1 \frown p_1 \frown \ldots \frown \ell_m \frown p_m$$

where the length of the representation of all ℓ_i and p_i is $\lceil \log n \rceil$ and $\frown$ stands for concatenation of sequences.

Now we define algorithm A using string S as advice. Any node v with label ℓ reads the unary representation of $\lceil \log n \rceil$ from S and searches for its label in S. Since the length of the representation of all ℓ_i and p_i is $\lceil \log n \rceil$, this can be easily done. If $\ell = \ell_i$ for some $i \in \{1, \ldots, r\}$, node v uses the port p_i first and then proceeds in the order of increasing port numbers to inform its remaining children. If $\ell \neq \ell_i$ for all $i \in \{1, \ldots, r\}$ (i.e. $\delta(v) \leq d$), node v informs its children in the order of increasing port numbers. Each node receives the whole string S from its parent when it is informed.

The sequence S is *wise* if, for every node v with label ℓ_i, port p_i leads to a child v_M of v such that $A(v_M) = \max\{A(w') : w' \text{ child of } v\}$. If, for any tree T, the root of T is provided with a wise advice sequence, then A is d-wise.

As the number m of nodes with down-degree $> d$ is in $O(n/d)$, a wise sequence S has length in $O(\frac{n \log n}{d})$. A wise sequence, given to the root, for every tree T, is the advice satisfying the lemma. □

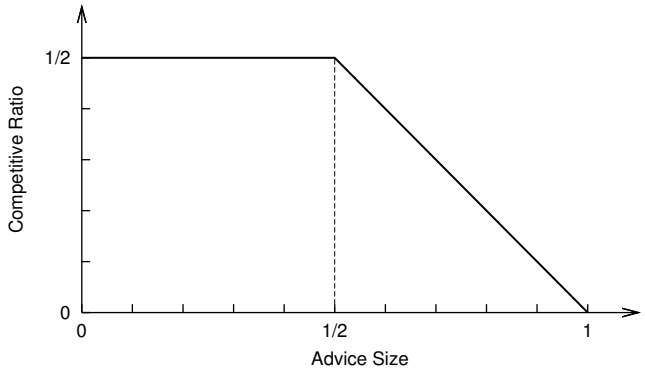

Figure 2.2: Approximate trade-off between the size of advice and competitive ratio (log-scale).

2.2.7 The Trade-off Curve

In this subsection we summarize our results concerning the trade-offs between the size of advice and the competitive ratio of broadcasting algorithms using it. Corollary 2.2.1 and Lemmas 2.2.3, 2.2.4, and 2.2.5 imply the following theorem.

Theorem 2.2.1 *If $q \leq \sqrt{n}$, competitive ratio $O(\sqrt{n})$ can be achieved for n-node trees, using q bits of advice; the best possible competitive ratio that can be achieved for n-node trees using q bits of advice is $\Omega\left(n^{\gamma}\right)$, for any constant $\gamma < 1/2$.*

If $q > \sqrt{n}$, competitive ratio $O\left(\frac{n \log^2 n}{q}\right)$ can be achieved for n-node trees, using q bits of advice; the best possible competitive ratio that can be achieved for n-node trees using q bits of advice is $\Omega\left(\frac{n^{1-\epsilon}}{q}\right)$, for any constant $\epsilon > 0$.

Hence, up to a n^{ϵ}-approximation factor, the best competitive ratio is $\Theta(\sqrt{n})$ for advice of size $q \leq \sqrt{n}$, and it is $\Theta\left(n/q\right)$ for advice of size $q > \sqrt{n}$.

For a fixed n, the trade-off between the size of advice and the competitive ratio of broadcasting algorithms in n-node trees is represented in Fig. 2.2 in logarithmic scale, with an approximation factor of n^{ϵ}, for any $\epsilon > 0$.

2.2.8 Information Cost of Optimal Broadcasting

In this subsection we study the size of advice needed to achieve time $\mathrm{opt}(T)$ of broadcasting on any tree T (i.e., broadcasting with competitive ratio 1). As follows from Corollary 2.2.2, $o(n)$ bits of advice are not sufficient to broadcast in optimal time in some trees. In fact, the class of (k, d)-trees requires advice of size $\Omega(n)$ even to achieve constant competitive ratio.

We first observe that $O(n \log n)$ bits of advice are always sufficient to perform broadcasting in optimal time.

Proposition 2.2.1 *There exists an algorithm A using $O(n \log n)$ bits of advice for n-node trees, and achieving time* $\mathrm{opt}(T)$ *for any n-node tree T.*

Proof: As follows from Cayley's theorem [39], $(n-2)\log n$ bits are enough to code all n-node trees, with nodes labeled by integers from 1 to n.

The Cayley's code of a given tree T is produced by concatenating to the label of the parent of the minimum labeled leaf ℓ the code produced by the tree T', obtained by pruning ℓ from T. The pruning process ends when the residual tree only contains 2 nodes.

In our setting, nodes need not only know the topology of the tree, but also the port numbers leading from a node to each of its children. This information can be added to the code, associating it with the label of the parent, thus doubling the length of the code. Notice that the parent relation in the Cayley's code assumes that the root of the tree is the node with label n, while the port information associated with the corresponding edge has to reflect the parent relation given by the root of the broadcasting tree, which not necessarily has label n.

If the advice given to the source is the Cayley's code of the tree, augmented by port information, then all nodes can obtain full knowledge of the tree from the advice string transmitted from the source, and consequently perform the optimal algorithm. □

It is natural to ask if the size $\Theta(n \log n)$ of advice is optimal. This would mean that performing optimal broadcasting in some trees requires an amount of information of the same order of magnitude as that needed to have complete knowledge of the tree. We do not believe that this is the case. Indeed we would like to propose the following conjecture.

Conjecture *The minimum size of advice needed to achieve optimal broadcasting time in all n-node trees is $\Theta(n)$.*

Intuitively, (k, d)-trees are difficult instances for the problem of broadcasting with advice. Indeed, all our lower bounds (including the one of Corollary 2.2.2) used this class of trees. The class of (k, d)-trees is the intersection of two well known classes of trees: caterpillars and d-ary trees. Caterpillars are trees in which all internal nodes form a line, while d-ary trees are trees whose internal nodes have down degree d. To support our conjecture, we show that these two classes of trees satisfy it.

Proposition 2.2.2

1. *$O(n)$ bits of advice suffice to broadcast in optimal time in n-node caterpillars.*
2. *$O(n)$ bits of advice suffice to broadcast in optimal time in n-node d-ary trees, for any d.*

Proof:

1. A broadcasting algorithm is optimal on any caterpillar, if every internal node informs first its only internal child. Hence it is enough to concatenate, on the advice string, the binary representations of port numbers leading to internal children. Each node of down degree larger than 1, knowing its down degree, is able to identify such information from the beginning of the string it obtains from its parent, then remove the corresponding bits, and pass the rest of the string to its internal child.

   It follows that $\sum_{v:\,\delta(v)>1} \lceil \log \delta(v) \rceil \leq \sum_{i=1}^{k} \lceil \log(n/k) \rceil \leq n$ (where $k < n/2$) bits of advice, are enough to achieve broadcasting in time $\mathrm{opt}(T)$ on any n-node caterpillar T.

2. First suppose that advice can be given to all nodes, not only to the source. Then, for each node v_i having $k \leq d$ internal children, we could provide a string s_i of length $k\lceil \log d \rceil$, indicating the permutation of internal children corresponding to the optimal order of transmissions (the order of leaf children is irrelevant). As there are at most n/d internal nodes, this would give a total of $S \leq (n/d)\lceil \log d \rceil \in O(n)$ bits of advice.

   Now return to our scenario where advice is given by the oracle only to the root of T, and relayed down the tree together with the source message. The advice given to the source is constructed as follows. The first $n + 1$ bits are 1's, followed by one 0, indicating the value of n. The following n bits are either 1 or 0, depending on whether the node with a given label i has internal children or not. The rest of the advice string is given by the concatenation of the strings s_i, separated by strings of $\lceil \log d \rceil$ zeroes.

Each node with label ℓ receives the full advice string from its parent. It reads the value n and it counts how many nodes with labels smaller than ℓ are provided with advice. This in turn indicates the (possibly empty) segment in the advice string corresponding to string s_ℓ, indicating the optimal permutation of internal children of node ℓ. The length of the whole string is at most $2n + 1 + 2S \in O(n)$.

□

2.2.9 Conclusion

We established trade-offs between the amount of advice given to the source and the best competitive ratio of a broadcasting algorithm in tree networks. Our results give the shape of the entire trade-off curve, but they are approximate: the gap between our upper and lower bounds on the best competitive ratio for a given size of advice is $O(n^\epsilon)$, for an arbitrarily small positive constant ϵ. A natural open problem is to tighten these bounds, ideally establishing the exact order of magnitude of the best competitive ratio for any size of advice. As far as the size of advice required for optimal broadcasting is concerned, we conjectured that this size is linear in the number of nodes and proved this property for two large subclasses of trees. Determining the status of our conjecture for all trees remains open.

Another open problem concerns the way in which advice is provided. In our case the entire advice is given to the root of the tree which is the source of broadcasting. While this is a natural way to proceed in the case of broadcasting, it would be interesting to study how the results change when portions of advice can be given by the oracle to all nodes, as was done in other works concerning network algorithms with advice. This way of giving advice, while more difficult to implement, could be potentially more efficient because it permits to save the bits identifying the node to which advice is given. However, we conjecture that the improvement of competitive ratio for a given size of advice, obtained in this way, would not be very significant.

2.3 UDG Radio Networks with Missing and Inaccurate Information

2.3.1 The Model and the Problem

In this section we assume a radio network consists of stations with identical transmitting and receiving capabililities. The model of communication is synchronous. In a given round each station can act either as a transmitter or as a receiver. We do not assume the availability of collision detection, thus a receiver is unable to distinguish a collision from silence.

The network is modeled as an **undirected graph** called a *unit disk graph* (UDG) whose nodes are the stations. These nodes are represented as points in the plane. Two nodes are joined by an edge if their Euclidean distance is at most 1. Such nodes are called *neighbors*. It is assumed that transmitters of all stations have equal power which enables them to transmit at Euclidean distance 1. It is also assumed that communication proceeds in a flat terrain without large obstacles. Hence the existence of an edge between two nodes indicates that transmissions of one of them can reach the other, i.e., these nodes can communicate directly. We refer to radio networks modeled by unit disk graphs as *UDG radio networks*.

The network topology is assumed to be unknown, namely, each node is unaware of the coordinates of any other node including its neighbors. Likewise, nodes do not know any bound on the size of the network or on its diameter. Such networks are often called *ad hoc* networks.

We assume that all stations are awake from round 1, and may contribute to the broadcasting process by transmitting control messages even before they heard the source message. In order for the broadcasting to be feasible, we assume that the unit disc graph underlying the UDG radio network is connected.

The above described scenario was adopted in [68] (under the name of the spontaneous wake up model) with two additional assumptions. It was assumed that the nodes are aware of (a linear lower bound on) the *density* d of the network, which is the minimum Euclidean distance between any two nodes, and that each node knows its **exact** position in the plane, i.e., is aware of the exact values of its Euclidean coordinates in a global coordinate system.

Each of these two assumptions may be difficult to satisfy in practical applications. If nodes of a radio network, e.g., sensors equipped with transmitters, are dynamically added to a network at unknown times and locations, it may be difficult or impossible to estimate the density of the network at any given time. On the other hand, reading Euclidean coordinates of a position, e.g., using a GPS, seems to be inherently prone to inaccuracies. Hence it is important to determine to what extent these two additional assumptions influence the time of broadcasting, and in particular, whether optimal broadcasting time changes if these two assumptions are removed.

We work in the above described model from [68] modified by removing the assumptions of density knowledge and of availability of exact positions of nodes. Instead, if the real position of the node is (x, y), we assume that the node perceives its position as being (x', y'), where (x', y') is chosen by an adversary. The distance between (x, y) and (x', y') is bounded by the *error margin* ϵ, a positive real parameter of the model. The point (x', y'), is called the *perceived position* of the node $v = (x, y)$ and is denoted by $P(v)$. It should be stressed that we do *not* assume the knowledge of ϵ either. All other characteristics of the model are as mentioned before, and as adopted in [68]. In particular, the topology, size and diameter of the network are unknown to nodes.

We consider only deterministic broadcasting algorithms and do not assume any central monitor of the broadcasting process. Thus the decision made by a node on whether to transmit or to receive in a given round, and what message to transmit, if any (some control messages can be transmitted on their own or be appended to the source message) is based exclusively on its perceived position and on the messages it heard so far. The *execution time* of a broadcasting algorithm in a given radio network is the smallest round number after which all nodes of the network have heard the source message and no other messages are sent.

It should be noted that our definition of the time complexity of a broadcasting algorithm is slightly different from the one adopted in [68] and the same as, e.g., in [43]. In [68] time was defined simply as the smallest round number t after which all nodes of the network have heard the source message. Hence the messages could be subsequently sent indefinitely by various nodes without any possibility of stopping and the algorithm would be still considered correct and running in a given time t. The difference between these two definitions is negligible if parameters of the network (in the case of [68] the diameter of the network and its density) are known to nodes. In this case the time of informing all nodes can be precomputed and the source can send a termination message after this time, without changing the order of magnitude of the total time of broadcasting. However if the diameter and/or the density are unknown, the difference between an algorithm that stops and one that does not is not trivial, and so is the difference between the two above definitions of execution time. We will see in the sequel that the requirement of total silence after the end of the algorithm is crucially used in our lower bound on the execution time. We consider the restriction to broadcasting algorithms that stop and the definition of time requiring silence after the last round to be more natural and useful: algorithms that do not stop require an external "help", e.g., user's intervention, to be implemented and thus cannot be considered to be autonomous algorithms. Hence we use this definition rather than that from [68]. However, it is an easy consequence of our result that the algorithm from [68] can be converted to a stopping algorithm and it will have the same order of magnitude of its running time according to our more demanding definition. In view of this, we refer to the time of the algorithm in [68], without distinguishing between the two definitions of running time of a broadcasting algorithm.

It was proved in [68] that the optimal broadcasting time (with each node knowing the density and its own exact position) is $\Theta\left(\min\left\{D+g^2, D\log g\right\}\right)$, where D is the diameter of the network (in the number of hops, which should not be confused with the Euclidean diameter of the set of points representing the stations, that is usually not the same), and g is the inverse of the density, called the *granularity* of the network (thus knowing d is equivalent to knowing g). The aim of the present section is to establish whether this optimal broadcasting time changes when the additional assumptions are removed.

2.3.2 Original Contribution

It turns out that the combination of missing and inaccurate information (unknown density and unknown error in perceiving positions) substantially changes the problem with respect to the easier scenario from [68]. The main challenge in our setting becomes fast broadcasting in sparse networks (with constant density), when optimal time is $O(D)$. (This was an easy task in the previous scenario.) A new difficulty comes from the fact that with unknown constant density d and unknown error margin ϵ possibly close to $d/2$, nodes with arbitrarily close perceived positions may be unable to communicate with each other. This invalidates election techniques used in [68]. Another component of difficulty is the stopping problem combined with ignorance of network parameters, in particular of the density and of the diameter. Not knowing these parameters makes it impossible to predict when the entire algorithm or its particular procedures will finish. Thus simple time-out conditions used in [68] are no more available and special care must be taken to explicitly stop the algorithm at the proper time.

Nevertheless, under our very weak scenario, we construct a broadcasting algorithm that maintains optimal time $O\left(\min\left\{D+g^2, D\log g\right\}\right)$ for all networks with at least 2 nodes, of diameter D and granularity $g = 1/d$ (a time previously obtained with exact positions and known density), assuming that each node perceives its position with error margin $\epsilon = \alpha d$, for any unknown constant $\alpha < \frac{1}{2}$. Somewhat surprisingly, the minimum time of an algorithm working correctly for *all* networks, and hence stopping if the source is alone, turns out to be $\Theta(D+g^2)$. Thus, the mere

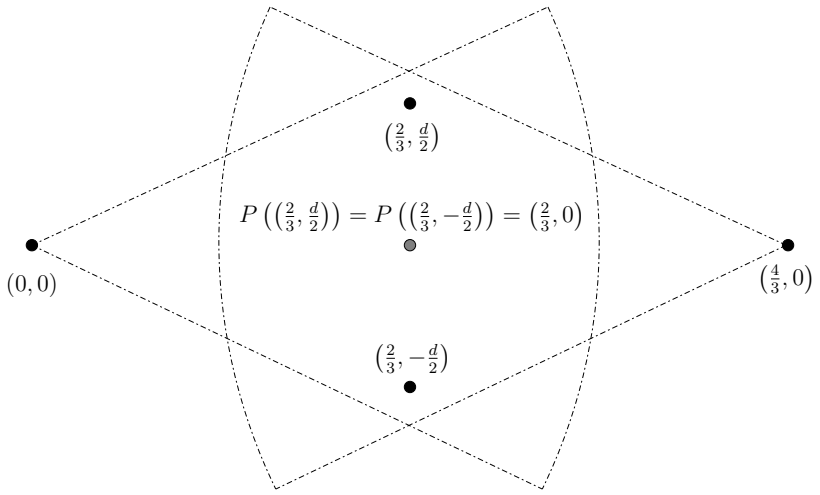

Figure 2.3: Impossibility of broadcast for $\epsilon \geq d/2$.

stopping requirement for the special case of the lonely source causes an exponential increase in broadcasting time, for networks of any density and any small diameter, e.g., polylogarithmic in g. Finally, we observe that the above upper bound on the error margin is necessary: indeed, if $\epsilon \geq \frac{1}{2}d$, then broadcasting turns out to be impossible in many networks.

2.3.3 Terminology and Preliminaries

We may assume that the (unknown) density d of the network is at most 1, otherwise all nodes would be isolated. The *granularity* of the network is $g = 1/d$. Throughout the section we assume that the error margin ϵ on the perception of node positions is at most αd, for some (unknown) constant $\alpha < 1/2$; this implies $\epsilon < 1/2$. We first observe that the above upper bound on ϵ is necessary.

Proposition 2.3.1 *For error margin $\epsilon \geq \frac{1}{2}d$ there exists a 4-node UDG radio network of density d in which broadcasting is impossible.*

Proof: Consider the UDG radio network based on the following four points: $(0,0)$, $(2/3, d/2)$, $(2/3, -d/2)$, $(4/3, 0)$, and let the point $(0,0)$ be the source (see Fig. 2.3). Suppose that $\epsilon \geq d/2$ and that the perceived positions of nodes are as follows: $P(0,0) = (0,0)$, $P(4/3,0) = (4/3,0)$, $P(2/3, d/2) = P(2/3, -d/2) = (2/3, 0)$. Consider any broadcasting algorithm A for this network. Since there exists an automorphism carrying node $(2/3, d/2)$ onto $(2/3, -d/2)$ and fixing the other nodes, and the perceived positions of nodes $(2/3, d/2)$ and $(2/3, -d/2)$ are identical, it follows by induction on the round t that the behavior of nodes $(2/3, d/2)$ and $(2/3, -d/2)$ will be identical in every round t: they will either both transmit or both receive in every round t. Since node $(4/3, 0)$ is adjacent only to these two nodes, it follows that it cannot get any message in any round t, and hence the algorithm A is incorrect. □

Algorithms proposed in [68] are based on three types of grids, whose definition we recall in the following.

Each of the three grids is composed of atomic squares with generic name *boxes*. The first grid is composed of boxes called *tiles*, of side length $d/\sqrt{2}$, the second of boxes called *blocks*, of side length $1/\sqrt{2}$, and the third one of boxes called *5-blocks*, of side length $5/\sqrt{2}$. All grids are aligned with the coordinate axes, each box includes its left side without the top endpoint and its bottom side without the right endpoint. Each grid has a box with the bottom left point with coordinates $(0,0)$.

Tiles are small enough to ensure that only one node can belong to a tile. Blocks are squares with diameter 1, i.e., the largest possible squares such that each pair of nodes in a square are able to communicate. 5-blocks are used to avoid collisions during communication: messages originating from central blocks of disjoint 5-blocks cannot cause collisions.

In our setting a node can decide if it is in a region of the plane only depending on its perceived position, which can be at distance ϵ from its real position.

We say that a node *inhabits* a given box if its perceived position belongs to this box. Two boxes are *potentially reachable* from one another, if they can be inhabited by a pair of nodes with real positions at distance at most 1. Two boxes are *reachable* from one another, if they are inhabited by such a pair of nodes.

If d can be as large as 1, any constant side length of a box is too large to ensure the property that nodes which inhabit the same box have real positions at distance at most 1, and hence that they can communicate. Indeed, let $d = 1$ and let ℓ be an arbitrary constant. Take ϵ such that $1/2 > \epsilon > (1-\ell)/2$ and consider a pair of nodes u and v at distance δ where $1 < \delta < \ell + 2\epsilon$. The distance between perceived positions of nodes u and v can be as small as $\delta - 2\epsilon < \ell$, hence they may inhabit a box of side length $\ell/\sqrt{2}$, but be unable to communicate. It follows that we cannot partition the plane into boxes guaranteeing full communication within a box. This motivates the design of separate algorithms for d below some threshold, when such a communication is possible, and for d above this threshold, in which case full communication within a box will not be needed.

Let $\Delta = \frac{\sqrt{2}-1}{2(\sqrt{2}+1)}$. We will take 2Δ as the threshold for d. We call a network *sparse* if $d \geq 2\Delta$ and *dense* otherwise.

Both in the algorithm for sparse networks and in algorithms for dense networks we will heavily use the concept of *multiplexing* of procedures. By multiplexing we mean that the execution of a procedure, described as a sequence of consecutive steps, will be interleaved with the execution of other procedures needed to complete the task. In general, only one step of each procedure is executed in a round robin fashion, unless explicitly stated. Divisions in blocks and (as we will see in the next subsection) assignment of colors are again used to interleave the execution of the same task in different rounds (depending on colors, on blocks or on a combination of both) in order to avoid collisions. Multiplexing will also be used on a higher level, in order to interleave the execution of different algorithms developed below. Indeed, as we are unaware of the value of d, we are unable to determine in advance if the network is sparse or dense, thus we have to run concurrently the algorithms for dense and sparse networks. This allows us to always complete the task in optimal time, by stopping slower algorithms after completion of the one that is the best for a given network.

2.3.4 Broadcasting in Sparse Networks

In this subsection we describe Algorithm `Color&Transmit`, working correctly for sparse networks.

We call *block* a box with side length 1. (Notice the difference from [68], where blocks had side length $1/\sqrt{2}$.) Blocks are used to build a grid in the same fashion as mentioned in Subsection 2.3.3. One more grid is composed of 5-*blocks*, i.e., boxes of side length 5. Nodes which inhabit a block must have real position within distance $\epsilon \leq 1/2$ from the block. It follows that any transmission made by a node inhabiting the central block of a 5-block can only be heard by nodes which inhabit the 5-block.

In sparse networks, $\gamma = \left\lceil \frac{\pi}{\sqrt{3}} \cdot \frac{\sqrt{2}+1}{\sqrt{2}-1} \cdot \left(2 + \frac{\sqrt{2}-1}{\sqrt{2}+1}\right)^2 \right\rceil$ is an upper bound on the number of nodes which can inhabit a block (we use $\frac{\pi}{\sqrt{12}}$ as the upper bound on the ratio of the sum of areas of pairwise disjoint circles of radius Δ in a square of side length $2 + 2\Delta$ to the area of this square).

We reserve a total of 25γ distinct colors. Nodes in block B_i of a 5-block, $1 \leq i \leq 25$, can be colored with colors from set C_i, where $|C_i| = \gamma$ and sets C_i are pairwise disjoint.

In our algorithm we will use a *grid refinement* process in order to perform various tasks. Nodes participating in grid refinement are inhabitants of a box. As it is impossible to guarantee full communication in a box with any constant side length, we use a distinguished node, called *witness*, which coordinates the process and determines the set of participants depending on the condition whether they are able to communicate with the witness or not.

The grid refinement process proceeds in phases. In the first phase, the whole box is divided in 4 square tiles, numbered from 1 to 4 proceeding left to right, top to bottom.

In successive phases, the side length of tiles is halved, thus quadrupling the number of tiles in the grid. Tiles are allotted rounds in a round robin fashion. A participating node with perceived position in the i-th tile, transmits during rounds allotted to its tile. Rounds allotted to the witness are interleaved with those of the tiles.

If only one participating node inhabits a tile, its transmissions will be heard by the witness (which thus learns the perceived position of the participating node). The witness sends a confirmation to the node immediately after.

If more than one participating node inhabits a tile, transmissions collide, thus the witness hears silence and does not send any confirmation.

At the end of each phase, it may be needed to check if the grid refinement has correctly terminated. If needed, such a check is performed as follows. First, one distinguished node u within communication range of the witness must be provided (such a node will be always explicitly defined whenever needed).

Once node u is fixed, it transmits together with all participating nodes that did not receive a confirmation during the current phase. The witness is thus able to distinguish whether the process is completed or not; if it hears the message from u, this means that the grid is fine enough to have at most one participating node in each tile and the process is complete. Otherwise, the grid is further refined by halving the side length of tiles, and a new phase begins.

Notice that only a constant number of phases are needed to complete the process because $\epsilon \leq \alpha d$, for some constant $\alpha < 1/2$, and hence the distance between perceived positions of distinct nodes is lower-bounded by a positive constant.

Upon completion of grid refinement, the witness and all participating nodes know the complete set of participants (participating nodes learn this set through confirmations of the witness).

Now we are able to describe Algorithm `Color&Transmit`. The dissemination of information from the source is based on a coloring of nodes in the network. The coloring algorithm defines a spanning tree of the network, whose height is in $O(D)$. The *parent* and *child* relations will always refer to this tree.

The coloring satisfies the following conditions: for every pair (v_1, v_2) of nodes having the same color, the set of children of v_1 in the spanning tree does not contain any neighbor of v_2 (i.e., v_2 is at distance greater than 1 from any children of v_1). Moreover, the parent of v_1 (respectively v_2) is not adjacent to v_2 (respectively v_1).

Siblings in the spanning tree have different colors and each node has a color different from the one of its parent. Notice that we are not enforcing the assignment of different colors to neighbors in the graph.

Once a coloring satisfying the above conditions is available, it is possible to ensure that transmissions from a parent (child) reach all its children (its parent) without collisions, by allotting distinguished rounds to nodes of different colors. The bounded height of the spanning tree ensures termination of broadcasting in time $O(D)$, provided that the total number of colors is bounded by a constant.

Procedure `Assign-Color`
Input: a pair of blocks B_i, B_j in a 5-block, and a color c in C_i.

The procedure assigns, in constant time, colors from set C_j to those nodes in B_j that are yet uncolored and have a neighbor with color c in block B_i, respecting the coloring conditions.

We say that a node is *out of the tree* if it does not know its parent. During the coloring process we mantain the invariant that nodes with the same color do not share neighbors out of the tree.

In the first round of Procedure `Assign-Color`, all nodes with color c in block B_i transmit. Next, out of the tree neighbors of each node w with color c in block B_i, inhabiting block B_j, start a grid refinement process with w as a witness. (Thus, many grid refinement processes may be simultaneously active.) Node w becomes the parent of participants of the grid refinement process.

In order to check termination of each grid refinement process, we need one more distinguished node u: this node is the parent of the corresponding node w.

Notice that each node w has a different parent, as siblings do not get the same color. Consider two nodes, w and w', with color c in block B_i. Let p be the parent of w and p' be the parent of w'. The coloring must satisfy that w is not a neighbor of p', thus transmissions made by p and p' cannot collide at w.

In constant time, all children of w, inhabiting block B_j (and w itself) know the full list L of perceived positions of children of w inhabiting block B_j. (Recall that children of w inhabiting block B_j may be unable to hear each other directly, so they rely on w to learn the list L.)

Subroutine `Conflict-Detection`
Input: a quadruple (B_i, B_j, c, c') where B_i and B_j are blocks and $c \in C_i$ and $c' \in C_j$ are colors.

The subroutine allows each node v in block B_j seeking color c', whose parent w in block B_i has color c, to distinguish between three possible outcomes:

1. node v can *win* color c';
2. node v can *lose* color c';
3. node v can *make a draw* on color c'.

All nodes seeking color c' are called *competitors*.

Subroutine `Conflict-Detection` works in 4 *consecutive* rounds (the usual interleaving between procedures is not respected for subroutine `Conflict-Detection` as interleaving different runs of the subroutine could cause unexpected behavior. As the number of rounds used by Subroutine `Conflict-Detection` is 4, the whole execution can be allotted a segment of 4 consecutive rounds, increasing the delay in the execution of other procedures by a constant factor only.).

In the first round of Subroutine `Conflict-Detection` each node v that has not yet lost color c' transmits, claiming color c'.

In the second round, all parents w of competing nodes transmit. At the same time, all non-competing nodes that did not hear any claim for color c' in the previous round transmit. If a competitor v heard the message from its parent w in the second round, no node v' sharing a non-competing neighbor with v is competing for color c'.

In the third round, all nodes w in B_i with color c transmit together with all non-competing nodes which know about a previous winner of color c'. If node v heard the message from its parent w in the third round, it means that none of its non-competing neighbors knows about a previous winner.

Node v wins color c', if it heard the message from its parent w in the second and third rounds. Node v loses color c', if it did not hear the message from its parent w in the third round (color c' was already won by a node sharing a non-competing neighbor with v). Node v makes a draw on color c' if it did not hear its parent w in the second round, but heard it in the third round.

Notice that children of v will be selected among out of the tree nodes, and thus are non-competing. Parents of nodes competing with v during the same execution of Procedure `Assign-Color`,

have the same color as the parent of v. It follows that they cannot be adjacent to v, as v would have been an out of the tree node when they were assigned a color, and thus either they or the parent of v would have lost the color.

In round 4, each competitor v announces the result: win, lose or draw, informing all its non-competing neighbors in case of victory and at least its parent w in other cases. ♣

Once the list L is known to a parent w with color c in block B_i, the first node v in lexicographic order of perceived positions in the list L starts competing for the first available color $c' \in C_j$ (a color is available if the parent does not know about a previous winner). This is done by calling Subroutine `Conflict-Detection`. If node v wins, Procedure `Assign-Color` removes v from list L and color c' from available colors for all remaining nodes in L. Then, the first node in list L starts competing for the first available color.

If node v loses, Procedure `Assign-Color` removes color c' from available colors for all nodes in L and v starts competing for the next available color.

If node v makes a draw, it still needs to compete for color c'. When a draw occurs, we say that nodes are in conflict; conflicts are resolved using Subroutine `Conflict-Resolution`.

Each competing node participating in Subroutine `Conflict-Resolution` will either win or lose the color it was competing for. Multiple runs of Subroutine `Conflict-Resolution` can be executed at the same time with the same block arguments if more than one parent w is present in block B_i. Resolution of conflicts is achieved by ensuring that each of the conflicting nodes becomes the only competitor in a run of Procedure `Conflict-Detection`, within constant time from the first draw. The latter is enough to ensure there is no draw.

This result is achieved by using a grid refinement process that delays successive executions of Subroutine `Conflict-Detection` by increasing amounts of time, as described below.

Subroutine `Conflict-Resolution`
Input: a quadruple (B_i, B_j, u, v), where $u \in B_i$, $v \in B_j$ and u is the parent of v.

Subroutine `Conflict-Resolution` proceeds in consecutive phases. Consider phase i for a conflicting node v. Let $p_i(v)$ be the number of the tile inhabited by node v in the i-th grid of the grid refinement process of block B_j. Clearly $1 \leq p_i(v) \leq 4^i$. Let S_v be the time when node v started participating in Procedure `Conflict-Resolution`. In step $S_v + w_i + 4^{i+1} + p_{i+1}(v)$, where $w_i = \sum_{j=1}^{i} 2 \cdot 4^i$ and $w_0 = 0$, node v starts participating in Procedure `Conflict-Detection` for the $(i+1)$-th time. If the outcome of this procedure for node v is a draw, the grid is further refined and a new phase begins. ♣

After resolution of a conflict, node v either wins or loses color c'. The actions of Procedure `Assign-Color` where already described in both these cases. ■

Lemma 2.3.1 *Procedure* `Conflict-Resolution` *operates in constant time.*

Proof: Consider nodes $v_1, \ldots, v_k$. Node v_i starts participating in Procedure `Conflict-Resolution` in time S_{v_i}, $S_{v_1} \leq S_{v_2} \leq \ldots \leq S_{v_k}$. Let δ_k be the sequence $(v_1, \ldots, v_k, S_{v_1}, \ldots, S_{v_k})$. Let $i(\delta_k)$ be the smallest phase such that $\left|p_{i(\delta_k)+1}(v_m) - p_{i(\delta_k)+1}(v_n)\right| > S_{v_m} - S_{v_n}$, for all $k \geq m > n \geq 1$. We first show that each node participating in Procedure `Conflict-Resolution` will be the only competitor in the $(i(\delta_k)+1)$-th run of Procedure `Conflict-Detection`.
By contradiction, assume that a node v_m is conflicting in its $(i(\delta_k)+1)$-th run of Procedure `Conflict-Detection`. Let v_n be one of the nodes that are conflicting with v_m.

The $(i(\delta_k)+1)$-th attempt of node v_m occurs in time $\rho = S_m + w_{i(\delta_k)} + 4^{i(\delta_k)+1} + p_{i(\delta_k)+1}(v_m)$. In order to have a conflict, we must have $S_n + w_j + 4^{j+1} + p_{j+1}(v_n) = \rho$ for some value j.

Consider three cases:

Case 1. $j < i(\delta_k)$.

$$\rho = S_{v_m} + w_{i(\delta_k)} + 4^{i(\delta_k)+1} + p_{i(\delta_k)+1}(v_m) > S_{v_m} + w_{i(\delta_k)} + 4^{i(\delta_k)+1} > S_{v_n} + w_{j+1} > S_{v_n} + w_j + 4^{j+1} + p_{j+1}(v_n)$$

as $j+1 \leq i(\delta_k)$ and $|S_{v_m} - S_{v_n}| < \left|p_{i(\delta_k)+1}(v_m) - p_{i(\delta_k)+1}(v_n)\right| < 4^{i(\delta_k)+1}$. Contradiction.

Case 2. $j = i(\delta_k)$.

$$\rho = S_{v_m} + w_{i(\delta_k)} + 4^{i(\delta_k)+1} + p_{i(\delta_k)+1}(v_m) = S_{v_n} + w_{i(\delta_k)} + 4^{i(\delta_k)+1} + p_{i(\delta_k)+1}(v_n),$$

hence $|S_{v_m} - S_{v_n}| = \left|p_{i(\delta_k)+1}(v_n) - p_{i(\delta_k)+1}(v_m)\right|$, which contradicts our assumption on $i(\delta_k)$.

Case 3. $j > i(\delta_k)$.

$$\rho = S_{v_m} + w_{i(\delta_k)} + 4^{i(\delta_k)+1} + p_{i(\delta_k)+1}(v_m) < S_{v_m} + w_{i(\delta_k)+1} < S_{v_n} + w_{i(\delta_k)+1} + 4^{i(\delta_k)+1} < S_{v_n} + w_j + 4^{j+1} + p_{j+1}(v_n),$$

as $|S_{v_m} - S_{v_n}| < \left|p_{i(\delta_k)+1}(v_m) - p_{i(\delta_k)+1}(v_n)\right| < 4^{i(\delta_k)+1}$. Contradiction.

Hence, in all cases, each node participating in Procedure `Conflict-Resolution` will be transmitting alone in its $(i(\delta_k)+1)$-th execution of Procedure `Conflict-Detection`.

Now we show that the completion time of Procedure `Conflict-Resolution` is bounded by a function $f(k)$, where k is the number of participants. This is enough in view of $k \leq \gamma$. We construct the values $f(k)$ by induction. Suppose that $f(k)$ has been already constructed.

Let $\sigma_k = S_{v_k} - S_{v_1}$. Since distances between perceived positions of nodes are lower-bounded by a positive constant, there exists an increasing function F such that $i(\delta_k) \leq F(\sigma_k)$. We now construct $f(k+1)$. We may assume $\sigma_{k+1} \leq f(k)$, for otherwise the procedure would end before the start of participation of node v_{k+1}, and hence we could take $f(k+1) = f(k)$. We already showed that the duration of Procedure `Conflict-Resolution` for nodes $v_1, \ldots, v_{k+1}$ is at most $\max_{j=1}^{k+1}(\sigma_j + w_{i(\delta_{k+1})} + 4^{i(\delta_{k+1})+1} + p_{i(\delta_{k+1})+1}(v_j))$. The latter is at most:

$$\sigma_{k+1} + 4^{i(\delta_{k+1})+2} \leq \sigma_{k+1} + 4^{F(\sigma_{k+1})+2} \leq f(k) + 4^{F(\sigma_{k+1})+2} \leq f(k) + 4^{F(f(k))+2}.$$

Hence we can take

$$f(k+1) = f(k) + 4^{F(f(k))+2}.$$

Thus the function f (depending only on the number k of participants) is constructed, which concludes the proof of the lemma. □

In order to complete Algorithm `Color&Transmit`, we need to describe how to initialize the whole process starting from the source s. The source is precolored with the first color avaiable for the block it inhabits. Using a grid refinement process, s can elect a distinguished node in its neighborhood in constant time and assign it a color. (If there is no neighbor of the source, i.e., the source is the only node in the network, the source transmits only once.) This distinguished node allows the source to check the correct termination of the grid refinement process. From now on, using Procedure `Assign-Color`, the whole network can be colored with 25γ colors. Notice that the time taken to assign a color to a node is constant. Once a node is colored, all its neighbors are colored after a constant time. It follows that the coloring ends in time $O(D)$ and thus the height of the induced spanning tree is in $O(D)$. After getting the source message, a node waits until the first round assigned to its color and transmits the message, informing all its children. Confirmation messages are sent back to the source along this tree, starting from the leaves, again using colors to avoid collisions on parent nodes. Each transmission is delayed by a constant time only, and the whole process is completed in time $O(D)$. Confirmation will be used in the main algorithms in Subection 2.3.6.

Theorem 2.3.1 *There exists a deterministic algorithm that completes broadcast in time $O(D)$ in any UDG radio network of unknown diameter D and unknown density $d \geq 2\Delta$.*

2.3.5 Broadcasting in Dense Networks

Our algorithm for sparse networks was based on the assumption that $d \geq 2\Delta$. Now we assume $d < 2\Delta$. We call Δ-*block* a box with side length $\frac{1-2\Delta}{\sqrt{2}}$. Δ-blocks are used to build a grid in the same fashion as before. One more grid is composed of 5Δ-*blocks* , i.e., boxes of side length $\frac{5}{\sqrt{2}}(1-2\Delta)$. Under the current assumption on d, the distance between the actual positions of two nodes in a Δ-block is < 1.

Lemma 2.3.2 *If $\epsilon < \Delta$, transmissions made by nodes inhabiting the central Δ-block of a 5Δ-block can only reach nodes inhabiting the 5Δ-block.*

Proof: The minimum distance between a node inhabiting a central Δ-block of a 5Δ-block $\mathcal{B}$ and a node not inhabiting $\mathcal{B}$ is at least $\sqrt{2}\,(1-2\Delta) - 2\epsilon > \sqrt{2}\,(1-2\Delta) - 2\Delta > \sqrt{2}\left(1 - \frac{\sqrt{2}-1}{\sqrt{2}+1}\right) - \frac{\sqrt{2}-1}{\sqrt{2}+1} = \frac{\sqrt{2}+1}{\sqrt{2}+1} = 1$. □

It follows that there are 24 Δ-blocks potentially reachable from any Δ-block.

An $O\left(D+g^2\right)$-time Algorithm

The algorithm `Elect&Transmit` proposed in [68] is based both on the perfect knowledge of positions and on the knowledge of the density of the network. This algorithm elects a pair of adjacent nodes called *ambassadors* for each pair of neighboring blocks in a preprocessing phase of time complexity $O(g^2)$. When all nodes know d, by fixing an order between blocks and between tiles in each block, it is possible to let each node transmit alone and reveal its position in $25g^2$ rounds. This is the first phase of preprocessing. In a second phase, the knowledge acquired by each node in the first phase is spread (in the same fashion) to all its neighbors.

As the time taken to complete transmissions in each block is identical and known in advance when d is known, this is enough to elect a pair of ambassadors for each pair of neighboring blocks. (Any fixed strategy, such as choosing the first pair in lexicographic order allows to reach the goal.) Once the pairs of ambassadors are defined, broadcasting can be completed in time $O(D)$.

In our setting d is unknown and we only have inaccurate knowledge of positions, hence it is impossible to use algorithm `Elect&Transmit` from [68]. In what follows, we provide a new algorithm working in time $O\left(D+g^2\right)$, called Algorithm `Dense-1`.

Recall, from Subsection 2.3.4, the description of the grid refinement process. Taking advantage of full communication within a Δ-block, we can mimic the grid refinement process in dense networks without having a predefined witness, as the role of the witness can be played by the first node that is able to transmit alone. Indeed, such a node is heard by all participants in the Δ-block; its perceived position is then appended to all the subsequent messages sent during the execution of the process, thus informing it of its role of witness as soon as a second node is able to transmit alone. In any Δ-block containing at least two nodes the grid refinement process ends in time $O(g^2)$ (the second node transmitting alone is used to check termination). Nodes that are alone in their Δ-block, would be involved in the grid refinement process forever, unless external help allows them to stop. (In our algorithm such help will be, of course, provided.)

In [68] the authors developed a procedure called `Conquer`. This procedure elects a pair of neighboring nodes in two blocks in time $O(\log g)$. The input of Procedure `Conquer` consists of two blocks, B_1 and B_2, and a node b_1 in B_1. Procedure `Conquer` can either be successful or unsuccessful, depending on whether a pair of adjacent nodes exists in the two blocks or not. Termination of Procedure `Conquer` was based on the assumption that d is known. Nodes elected by procedure `Conquer` can be used as ambassadors to spread information from one block to another, but in our setting we need to take additional care in order to guarantee termination.

Logarithmic time is achieved in Procedure `Conquer` thanks to the ability of electing a neighbor of a given node c in a region R (of diameter at most 1) in time $O(\log g)$. This result is based on the procedure `Echo` from [118]. Procedure `Echo` allows all nodes at distance one from c and from all points in R to know if there are 0, 1 or more than one nodes in R, and can be used to perform elections of a node in a set in logarithmic time. The latter is achieved by using a halving process exponentially decreasing the area of the region where the node can be elected. The knowledge of d allows the node c to precompute the duration of an unsuccessful election (i.e., an attempt to elect a node from the empty set). In our setting this is impossible, thus we need to avoid involving nodes in unsuccessful halving processes while electing ambassadors for neighboring Δ-blocks.

In our algorithm, election of ambassadors is performed using Procedure `Safe-Conquer` detailed below.

Procedure `Safe-Conquer`
Input: two Δ-blocks, B_1 and B_2, a node a in B_1, and a node b adjacent to a.

Procedure `Safe-Conquer` elects a pair (u, v) of adjacent nodes such that $u \in B_1$ and $v \in B_2$ in time $O(\log g)$ whenever such a pair exists, otherwise it *stops* in constant time. We say that Procedure `Safe-Conquer` is successful if such a pair is elected, otherwise it is unsuccessful.

In the first round of execution of Procedure `Safe-Conquer`, all nodes in B_2 transmit together with b. If a cannot hear the message from b, it means that a node b_2 in B_2 exists such that b_2 is adjacent to a. If this is the case, a node v adjacent to a in B_2 is elected, completing the pair (a, v).

If no such a node b_2 exists, we need to verify if a pair (b_1, b_2) of adjacent nodes exists such that $b_1 \in B_1$ and $b_2 \in B_2$. This is done as follows: first a transmits a control message together with all nodes in B_2. Then all those nodes in B_1 which did not hear a, transmit. Simultaneously b transmits. Notice that node b can be inside B_1. If this is the case, it has to declare when transmitting if it heard a in the previous round. If a does not hear b or if the message from b states that it did not hear a previously (and thus it can communicate with a node in B_2), it means that there exists a pair (b_1, b_2) that we were searching for. Therefore we can apply Procedure `Conquer` (using Δ-blocks instead of blocks) and successfully elect such a pair (u, v). If the pair (b_1, b_2) does not exist, the two Δ-blocks are unable to communicate and no pair will be elected. In this case Procedure `Safe-Conquer` stops after 3 steps. ■

Now we have all the tools we need to describe Algorithm `Dense-1`.

In the first round, the source transmits alone. The source will not transmit any other message unless it hears a message from another node, allowing the protocol to end in time 1 when the whole network consists only of the source.

Starting from round 2, all Δ-blocks start the grid refinement process. Let B be a Δ-block in which the process ends, and let a and b be the first two nodes that transmitted alone in B. For any such block, Procedure `Safe-Conquer` is applied using as parameters B, B', a, and b, for every Δ-block B' potentially reachable from B (there are at most 24 such Δ-blocks).

As soon as the source hears a message (which happens after time $O\left(g^2\right)$ in networks with more than one node), it starts participating in two tasks. The first task is the grid refinement in its own Δ-block. The second task is the election of a node b among those informed during the first round, based on halving. This election is performed for each of the 25 Δ-blocks potentially reachable from the source, including its own Δ-block (using multiplexing). Election will be successful in one of these Δ-blocks and it will be completed in time $O(\log g)$. Let S be the Δ-block inhabited by the source s. As soon as node b is elected, Procedure `Safe-Conquer` is applied using as parameters S, T, s and b, for any Δ-block T potentially reachable from S (again, there are at most 24 such Δ-blocks).

For any Δ-block with at least 2 nodes (and for the Δ-block inhabited by the source), election of all ambassadors is completed in time $O\left(g^2 + \log g\right) = O\left(g^2\right)$.

Whenever a pair of ambassadors (u, v) for a pair of blocks (B, B') is elected, if B' is a Δ-block where grid refinement is still running, node v verifies if it is alone in B'. This is done by reserving a round where each node in B', except v, transmits together with u. If v can hear u, it is alone in B' and it stops executing grid refinement, otherwise other nodes are present and grid refinement will eventually terminate. It follows that after $O\left(g^2\right)$ time, the only Δ-blocks that are still running grid refinement are those Δ-blocks containing only one node that have no neighbors in Δ-blocks with 2 or more nodes. For any such Δ-block B, the election of ambassadors for the 24 pairs of blocks (B, B'), where B' is potentially reachable from B, can be completed in constant time using Procedure `Safe-Conquer`. In each case, the parameters of this procedure will be B, B', the unique node u inhabiting B, and the node v that first informed u.

The source message is then passed through ambassadors. Let (u, v) be a pair of ambassadors for Δ-blocks B, B'. If v is newly informed by u, it informs B' and informs u that B' has been newly informed by B. In this case we say that (u, v) is the *informing couple* of B'. Otherwise v does not transmit. A Δ-block is a leaf if it does not newly inform any other Δ-block. Confirmation of broadcast completion proceeds from leaf Δ-blocks to the source, again using ambassadors. Confirmation will be used in the construction of the main algorithms in Subection 2.3.6.

Theorem 2.3.2 *There exists a deterministic algorithm that completes broadcast in time $O\left(D + g^2\right)$ in any UDG radio network of unknown diameter D and unknown density $d = 1/g < 2\Delta$.*

An $O(D \log g)$-time Algorithm

In order to obtain an algorithm running in time $O(D \log g)$ we restrict attention to networks with at least two nodes. In Subection we will show that this restriction is necessary.

Algorithm `Dense-2`, is designed for networks of size at least 2 and works in time $O(D \log g)$. In such networks, we are guaranteed that there exists a node b adjacent to the source s, thus we can elect such a node b in logarithmic time in the same fashion as we did in Algorithm `Dense-1`, without waiting for the source to receive any message. Procedure `Safe-Conquer` is then used to elect pairs of ambassadors for any pair of Δ-blocks (B, B'), where B is inhabited by the source and B' is reachable from B. In general, fix a Δ-block B_1 not reachable from B. Consider the informing couple (u, v) of ambassadors for B_1. Nodes u and v are then used as parameters of Procedure `Safe-Conquer` for any Δ-block B_2 potentially reachable from B_1.

Election of ambassadors and spreading of the source message proceeds by a wave originating from the Δ-block inhabited by the source. Confirmation is done as previously. Since each election takes time $O(\log g)$, broadcasting is completed in time $O(D \log g)$.

Theorem 2.3.3 *There exists a deterministic algorithm that completes broadcast in time $O(D \log g)$ in any UDG radio network with at least two nodes, unknown diameter D and unknown density $d = 1/g < 2\Delta$.*

2.3.6 The Main Algorithms

Algorithms `Dense-1` and `Dense-2` developed for dense networks, may fail when applied on a sparse network, as the assumption of having full communication inside a Δ-block may not hold. On the other hand, Algorithm `Color&Transmit`, developed for sparse networks, can fail on dense networks, as it can run out of colors when there are more than γ nodes in a block. In order to develop an algorithm that is suitable for all networks, we multiplex the execution of algorithms conceived for dense and for sparse networks and use confirmation in order to stop the execution of the slower ones after the completion of the fastest. (Notice that such stopping is crucial because grid refinement for dense networks could run forever on sparse networks.) The only nodes that send confirmation

back to the source in algorithms for dense networks are the ambassadors. We need to ensure that whenever an algorithm for dense networks fails, at least one of the ambassadors becomes aware of the error, thus not sending confirmation back to the source. On the other hand, in algorithm `Color&Transmit` all nodes send confirmation, and thus failures are readily detected.

Procedure `Error-Detection`

Input: a Δ-block B.

Let (u, v) be the informing couple of B. (If B is the Δ-block inhabited by the source s, the informing couple is replaced by the pair (b, s), where b is the elected neighbor of s.) Procedure `Error-Detection` flags an error whenever a node at distance (in hops) at most 2 from v is not informed.

In the first round, v transmits together with all uninformed nodes. Let x be an informed node adjacent to v. x hears v in the first round, if and only if, it has no neighbors that are not informed. In the second round of the procedure, u transmits together with all neighbors of v that did not hear v in the previous round. The ambassador v flags an error (and does not send confirmation), if it does not hear u in the second round. ■

Procedure `Error-Detection` is called by algorithms `Dense-1` and `Dense-2` in any Δ-block B as soon as the following two conditions are satisfied:

1. confirmation from all Δ-blocks newly informed by B were obtained;
2. if B_1 is a Δ-block reachable from B and B_2 is a Δ-block reachable from B_1, then B_2 was informed.

(In particular, Procedure `Error-Detection` is not called in the singleton network.)

Lemma 2.3.3 *Let A be either Algorithm* `Dense-1` *or* `Dense-2`*. If A fails to inform all nodes, then A does not provide confirmation to the source.*

Proof: In order for A to fail, there must exist a node x that never received the source message. First suppose that a neighbor y of x is informed: it follows that there exists an ambassador v that informed y, and hence the distance in hops between x and v is equal to 2 and an error is flagged in the Δ-block inhabited by v. If all neighbors of x are not informed, let x' be the first node on a shortest path between x and an informed node. Repeat the reasoning replacing x with x'. As the graph is connected and the source is informed, such a shortest path always exists. The lemma follows by induction on the length of this shortest path. □

Lemma 2.3.4 *Let A be either Algorithm* `Dense-1` *or* `Dense-2`*. If A is executed in a dense network, then A provides confirmation to the source.*

Proof: Let B be the input of Procedure `Error-Detection`, $v \in B$ the ambassador which informed B, and x an uninformed node. Let B_2 be the Δ-block inhabited by x. B_2 must be unreachable from B and from any Δ-block B_1 which is reachable from B, as otherwise x would have been already informed. It follows that none of the neighbors of v has x as a neighbor, hence Procedure `Error-Detection` does not flag an error on B. This proves the lemma. □

We propose two main algorithms: Algorithm `Universal Broadcast` that works for all networks, and Algorithm `Company-Aware Broadcast` that works for all networks of size at least 2. Algorithm `Universal Broadcast` consists of multiplexing Algorithm `Color&Transmit` with Algorithm `Dense-1`, while Algorithm `Company-Aware Broadcast` consists of multiplexing algorithms `Color&Transmit`, `Dense-1`, and `Dense-2`.

The running time of Algorithm `Universal Broadcast` is $O\left(D+g^2\right)$. Indeed, if confirmation is provided to the source by Algorithm `Dense-1`, the source can stop the execution of Algorithm `Color&Transmit`, in time $O(D)$, using the already elected ambassadors to spread the termination message to the whole network. On the other hand, if confirmation is provided to the source by Algorithm `Color&Transmit`, the execution of Algorithm `Dense-1` can be stopped, again in time $O(D)$, using the already defined coloring to avoid collisions while spreading the termination message from the source to the whole network. Algorithm `Universal Broadcast` is successful on any network, and its running time is always bounded by the minimum of the running time of Algorithm `Color&Transmit` and Algorithm `Dense-1`, thus it is $O\left(D+g^2\right)$. Notice that, if the network contains only the source, the source will never receive any confirmation. Nevertheless, neither Algorithm `Color&Transmit` nor Algorithm `Dense-1` would use the source to transmit more than once, thus allowing the combined algorithm to end in constant time. Hence we have the following result, which will be proved optimal in Subection 2.3.7 by providing a matching lower bound.

Theorem 2.3.4 *There exists a deterministic algorithm that completes broadcast in time $O\left(D+g^2\right)$ in any UDG radio network of unknown diameter D and unknown density $d=1/g$.*

If we neglect networks with only one node, Algorithm `Company-Aware Broadcast` can be used. Stopping of the slower component algorithms by the fastest is done as previously, thus guaranteeing time $O\left(\min\left(D+g^2, D\log g\right)\right)$. However, Algorithm `Company-Aware Broadcast` runs forever if executed on a network containing the source only, hence it is not a correct algorithm for all networks. Since the lower bound $\Omega\left(\min\left(D+g^2, D\log g\right)\right)$ from [68] also holds in our case, Algorithm `Company-Aware Broadcast` is optimal whenever it is correct. Hence we have the following result.

Theorem 2.3.5 *The optimal time of a broadcasting algorithm working correctly on all UDG radio networks with at least two nodes, unspecified diameter D and unspecified density $d=1/g$, is $\Theta\left(\min\left(D+g^2, D\log g\right)\right)$.*

2.3.7 Lower Bound on Universal Broadcasting Time

In this subsection we assume that the source does not have any information whether it is the only node in the network or not. We want to establish optimal time of a broadcasting algorithm working correctly for *all* networks. In particular, this algorithm must stop after some fixed time when the source is the only node in the network. We show that this forces a lower bound $\Omega(g^2)$ on broadcasting time for some UDG radio networks of constant diameter and from there we derive the lower bound $\Omega(D+g^2)$ on broadcasting time for the class of UDG radio networks with unknown diameter D and unknown diameter d. This matches the time $O(D+g^2)$ of Algorithm `Universal Broadcast` from Subection 2.3.6, thus establishing $\Theta(D+g^2)$ as optimal broadcasting time for the class of arbitrary networks.

Consider two squares: A with corners $(\frac{3}{5},-\frac{1}{10}),(\frac{3}{5},\frac{1}{10}),(\frac{4}{5},-\frac{1}{10}),(\frac{4}{5},\frac{1}{10})$ and B with corners $(\frac{6}{5},-\frac{1}{10}),(\frac{6}{5},\frac{1}{10}),(\frac{7}{5},-\frac{1}{10}),(\frac{7}{5},\frac{1}{10})$ (see Fig. 2.4).

For any positive constant d, consider a d-grid inside each of those squares defined as the set of points in A (respectively, in B), including the lower-left corner and forming a grid with square side d. Let S be the d-grid in A and T the d-grid in B. Note that, for any d, the size $\sigma(d)$ of both sets S and T is the same and it is $\Omega(g^2)$. For any $S'\subseteq S$ and any $T'\subseteq T$ define a UDG radio network $N(S',T')$ whose set of stations is $\{(0,0)\}\cup S'\cup T'$ and whose source is the point $(0,0)$. Let $\mathcal{N}$ be the class of all networks $N(S',T')$, for all $d=1/g$, where g is an integer greater than 4, and for all $S'\subseteq S$ and $T'\subseteq T$. Note that networks of the class $\mathcal{N}$ have a simple structure: they are composed of two cliques, one on the set $\{(0,0)\}\cup S'$ and the other on the set T', with all possible edges between sets S' and T'. A network obtained for a given d has density d and diameter 2.

In the proof of our result we will use the following lemma (cf. Procedure Modify from [120]).

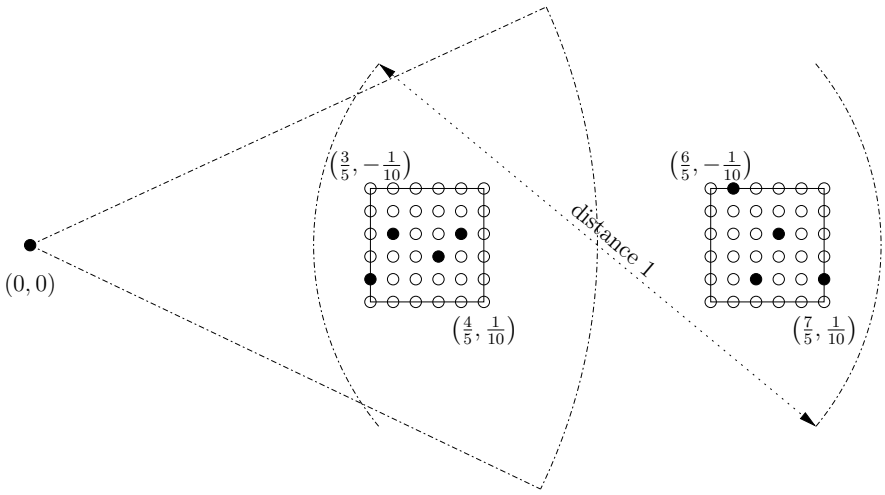

Figure 2.4: A network of the class $\mathcal{N}$.

Lemma 2.3.5 *For any sequence of sets $A_1, A_2, \ldots, A_k$ of subsets of V, where $|V| = n$, there exists a set $R \subseteq V$ such that $|R| \geq n - k$ and $|R \cap A_j| \neq 1$, for all $j \leq k$.*

Proof: Define recursively the following sequence of sets:
$R_1 = V$;
for any $i \leq k$, $S_i = \{x \in R_i : |R_i \cap A_j| = 1 \text{ for some } j \leq k\}$;
$R_{i+1} = R_i \setminus S_i$.
Now the set $R = R_{k+1}$ has the desired properties. □

Theorem 2.3.6 *For any broadcasting algorithm A working correctly on all networks of the class $\mathcal{N}$, and for every $g > 4$, there exists a network of density $d = 1/g$ in the class $\mathcal{N}$, for which the algorithm A uses time $\Omega(g^2)$.*

Proof: Fix a broadcasting algorithm A working correctly on all networks of the class $\mathcal{N}$. For some constant $\alpha > 0$ and for all g large enough we will construct a network $N(S^*, T^*)$ of density $d = 1/g$ in the class $\mathcal{N}$ for which the algorithm A uses time at least αg^2. (This is sufficient, as for smaller g we can simply take the network $N(S, T)$.) First notice that the network O consisting only of the source $(0, 0)$ is a member of the class $\mathcal{N}$. Hence algorithm A must work correctly on network O, which implies that after c rounds, for some positive constant integer c, the source must stop transmitting messages, if it is in the network O. Now suppose that the source is in some other network of the class $\mathcal{N}$. If during the first c rounds it does not hear any message, it cannot transmit any message in subsequent rounds, until it gets another message, because after c rounds without hearing any message it cannot distinguish if it is in the network O or in the other network and hence (assuming that it did not hear any messages later on) it must behave like it did in the network O. Let π be the pattern of behavior of the source during the first c rounds, if it does not hear any message during these c rounds. π can be represented as a binary sequence of length c, where 1 in the ith position indicates that the source transmits, and 0 that it does not. (The content of the transmitted message is irrelevant for our considerations.)

Consider the set T from the definition of the class $\mathcal{N}$ and suppose that g is large enough to guarantee $|T| > c + \lceil |T|/2^c \rceil - 2$. Let $t = c + \lceil |T|/2^c \rceil - 2$. Let $A_1, \ldots, A_t$ be the set of nodes from T that transmit in rounds $1, \ldots, t$, respectively, assuming that they did not hear any message in any of the rounds $1, \ldots, t$. Let T^* be the set R from Lemma 2.3.5 obtained by taking $k = t$ and $V = T$. The set T^* is non-empty and has the property that in any network $N(Y, T^*)$, if nodes from

T^* do not hear any messages from Y in rounds $1, \ldots, t$ then, in each of those rounds, either no node from T^* transmits (call such a round T^*-*silent*), or at least two nodes from T^* transmit (call such a round T^*-*loud*). The construction of T^* is completed.

Now we proceed with the (somewhat more complicated) construction of the set S^*. Consider the set S from the definition of the class $\mathcal{N}$ and suppose that g is large enough to guarantee $|S| \geq 2^c+1$. Let $S_0 = S$ and let S'_0 be the set of those nodes in S_0 that transmit in the first round, according to algorithm A. If $|S'_0| \geq |S_0|/2$ then let $S_1 = S'_0$, otherwise, let $S_1 = S_0 \setminus S'_0$. In the first case call round 1 *dense*, in the second case call it *sparse*. Now suppose that the outcome of round 1 was the following for nodes in S_1: if round 1 was dense then nodes in S_1 did not hear anything; if round 1 was sparse and either $\pi(1) = 0$ or round 1 was T^*-loud then nodes in S_1 did not hear anything; if round 1 was sparse, $\pi(1) = 1$ and round 1 was T^*-silent then nodes in S_1 heard a message from the source. Let S'_1 be the set of nodes in S_1 that transmit in round 2 according to algorithm A *under the above assumption*. If $|S'_1| \geq |S_1|/2$ (round 2 was dense) then let $S_2 = S'_1$, otherwise (round 2 was sparse), let $S_2 = S_1 \setminus S'_1$. Continuing in this way we define by induction sets $S_0, S_1, \ldots, S_c$, at each step taking as S_{i+1} the set of those nodes in the previously defined set S_i that exhibited the majority behavior under the previously assumed history, (which is composed of sequences of messages – including the empty message – assumed to be heard by each node in consecutive rounds). The assumptions on the history before round i are satisfied in any network $N(S^*, T^*)$ for which $S^* \subseteq S_i$ and $|S^*| \geq 2$. Indeed, in any such network, in dense rounds prior to i at least two nodes from S^* would transmit and in sparse rounds prior to i no node from S^* would transmit. Consequently the source and the nodes in T^* would not hear any message from S^* and would behave as described before.

Now let $\bar{S} = S_c$. By our construction, the set $\bar{S}$ has size at least $z = \lceil |S|/2^c \rceil \geq 2$. It has the property that in any network $N(S^*, T^*)$ for which $S^* \subseteq \bar{S}$ and $|S^*| \geq 2$, no node can hear any message from a node in S^* in or before round c. This is due to the fact that in all these rounds the number of transmitting nodes from S^* is either 0 or at least 2. In particular, in any such network, the source cannot hear any message in the first c rounds. By the definition of c, in any such network the source will not transmit after round c unless it hears a message.

We can now finish the construction of S^*. Let $k = z - 2$ and let $B_1, \ldots, B_k$ be sets of nodes in $\bar{S}$ transmitting, respectively, in rounds $c+1, c+2, \ldots c+k$, provided that they did not hear any message after round c. Let S^* be the set R from Lemma 2.3.5 obtained by taking $V = \bar{S}$ and substituting sets B_i for A_i. This concludes the entire construction. It is now enough to prove that the network $N(S^*, T^*)$ has the desired properties.

First note that $S^* \subseteq \bar{S}$ and $|S^*| \geq 2$. As remarked before, the source cannot hear any message in the first c rounds. Hence it will not transmit after round c unless it hears a message. Moreover, in the first c rounds, nodes in T^* cannot hear any message from a node in S. Let us now consider rounds $c+1, c+2, \ldots c+z-2$. By induction on the round number the following invariant can be shown to hold in all these rounds:

- the source does not transmit;
- no node of the network hears any message.

We conclude that nodes in T^* cannot hear the source message in any of the rounds $1, \ldots, c+z-2$. Consequently, broadcasting in the network $N(S^*, T^*)$ takes time at least $c+z-1$, because T^* is non-empty. This holds for g large enough to guarantee both $\sigma(d) > c + \lceil \sigma(d)/2^c \rceil - 2$ and $\sigma(d) \geq 2^c + 1$ (recall that $|S| = |T| = \sigma(d)$). Now take $\beta = 2^{-(c+1)}$. Since, for sufficiently large g, we have $c + z - 1 = c + \lceil \sigma(d)/2^c \rceil - 1 \geq \beta \cdot \sigma(d)$, and $\sigma(d)$ is in $\Omega(g^2)$, we conclude that there exists a constant $\alpha > 0$ such that, for sufficiently large g, algorithm A uses time at least αg^2 to broadcast in network $N(S^*, T^*)$. □

It is easy to generalize the above lower bound to UDG radio networks of arbitrary diameter D and density d. Instead of networks of the class $\mathcal{N}$ that consist of a source $(0,0)$ and d-grids in squares A and B, we take the source $(0,0)$ followed by $D-1$ squares of side $1/5$, arranged in a line with distances $3/5$ between consecutive square centers. In each of the squares we insert d-grids as before. It is easy to see that the $\Omega(g^2)$ lower bound can be proved as above, while the diameter D is a trivial lower bound on broadcasting time. Hence we get the following corollary.

Corollary 2.3.1 *For any broadcasting algorithm A working correctly on all UDG radio networks, there exist networks of diameter D and density $d = 1/g$, for arbitrary D and g, for which algorithm A uses time $\Omega(D + g^2)$.*

It should be noted that the above lower bound remains true even when nodes know exactly their positions, i.e., for error margin 0. Theorem 2.3.4 and Corollary 2.3.1 imply the following result.

Corollary 2.3.2 *The optimal time of a broadcasting algorithm working correctly on all UDG radio networks with unknown diameter D and unknown density $d = 1/g$ is $\Theta(D + g^2)$.*

2.3.8 Conclusion

In this section we studied broadcasting time in radio networks, modeled as Unit Disc Graphs, under a more realistic set of assumptions with respect to the ones previously adopded in the literature.

Two impractical assumptions have been removed:

- the knowledge of the density;
- the ability of nodes to perceive their exact position in the plane.

Our work shows that optimal broadcasting time does not change in this modified setting, but new difficulties arose, in particular for sparse networks.

We also introduced more strict requirements for termination of the algorithms. The requirement of silence, combined with ignorance of network parameters like the density and the diameter, can cause an exponential gap in optimal broadcasting time for all networks of small diameter, unless we allow our algorithms to run forever in the singleton network.

2.4 Impact of Information on Asynchronous Radio Broadcasting

2.4.1 The Model and the Problem

A radio network consists of stations with transmitting and receiving capabililities. The network is modeled as a **directed graph** with a distinguished node called the *source*. Each node has a distinct identity (label) which is a positive integer. If there is a directed edge from u to v, node v is called an *out-neighbor* of u and u is called an *in-neighbor* of v.

At some time t a node may send a message to all of its out-neighbors. It is assumed that this message is delivered to all the out-neighbors simultaneously at some time $t' > t$ decided by an adversary that models unpredictable asynchronous behavior of the network. The only constraint (cf. [46, 145]) is that the adversary cannot collate messages coming from the same node, i.e., two distinct messages sent by the same node have to be delivered at different times. We consider two types of asynchronous adversaries. The *strong* adversary, called the *node adversary* in [46], may choose an arbitrary delay $t' - t$ between sending and delivery, possibly different for every message. The *weak* adversary chooses an arbitrary delay for a given node (possibly different delays for different nodes), but must use this delay for all messages sent by this node during the protocol execution. The motivation for both adversaries is similar and follows the one given in [46]. Nodes of a radio network execute a communication protocol while concurrently performing other computation tasks. When a message arrives at a node, it is stored (prepared for transmission) and subsequently transmitted by it, the (unknown) delay between these actions being decided by the adversary; storing for transmission corresponds to sending and actual transmission corresponds to simultaneous delivery to all out-neighbors (at short distances between nodes the travel time of the message is negligible). The delay between storing and transmitting (in our terminology, between sending and delivery) depends on how busy the node is with other concurrently performed tasks. The strong adversary models the situation when the task load of nodes may vary during the execution of a broadcast protocol, and thus delay may vary from message to message even for the same node. The weak adversary models the assumption of a constant occupation load of each node during the communication process: some nodes may be more busy than others but the delay for a given node is constant.

At time t', a message is *heard*, i.e., received successfully by a node, if and only if, a message from exactly one of its in-neighbors is delivered at this time. If messages from two in-neighbors v and v' of u are delivered simultaneously at time t', we say that a *collision* occurs at u. Similarly as in most of the literature concerning algorithmic aspects of radio communication, we assume that in this case u does not hear anything at time t', i.e., we assume that a node cannot distinguish collision from silence.

While in general the network is modeled as an arbitrary directed graph, we also consider two smaller classes of networks. The first is the one of symmetric networks, modeled by symmetric directed graphs, or equivalently by undirected graphs. The second, still smaller class of networks is the one of UDG networks. In the case of UDG networks, each node knows its Euclidean coordinates in the plane. These coordinates also play the role of the label (similarly as, e.g., in [68, 69] and in the previous section, nodes in UDG networks are not equipped with integer identities).

2.4.2 Centralized vs. Ad Hoc Broadcasting

We assume that only stations that have already received the source message can send messages (conditional wake up model). In order for the broadcasting to be feasible, we assume that there is a directed path from the source to any other node. For symmetric networks this is equivalent to connectivity. As in the rest of this chapter, we consider only deterministic broadcasting algorithms.

Two alternative assumptions are made in the literature concerning broadcasting algorithms. It is either assumed that the topology of the underlying graph is known to all nodes, in which case nodes

can simulate the behavior of a central monitor scheduling transmissions (*centralized broadcasting*), or it is assumed that the network topology is unknown to nodes (*ad hoc broadcasting*). Moreover, in the latter case, some crucial parameters of the network, such as the number n of nodes, may be known or unknown to nodes. In the case of UDG radio networks, an important parameter is the density d of the network, as widely discussed in the previous section. We will see how information about the topology of the network and knowledge of its parameters influence the efficiency of broadcasting protocols. In particular, for UDG networks, optimal work of broadcasting protocols may depend on the granularity g of the network, which we recall is defined as the inverse of the density.

2.4.3 Adaptive vs. Oblivious Protocols

We consider two kinds of broadcasting protocols: oblivious and adaptive. In an oblivious protocol every node has to send all its messages as soon as it is woken up by the source message. More precisely, a node has to commit to a non-negative integer representing the number of messages it will send during the broadcasting process, prior to the execution of the protocol. This number may depend only on the label of the node or on its position in the case of UDG networks. (In [46] only oblivious protocols were considered.) By contrast, an adaptive protocol is more powerful, as a node can decide on the number and content of messages it sends, depending on its history, i.e., depending on the sequence of messages received so far. Hence, while the total number of messages sent by an oblivious protocol is the same for each of its executions, for an adaptive protocol this number may differ depending on the behavior of the adversary.

We define the *work* of a broadcasting protocol as the worst-case total number of messages sent until all nodes are informed. The worst case is taken over all possible behaviors of an asynchronous adversary under consideration. Work is a natural measure of complexity of an asynchronous radio broadcast protocol. It was introduced in [46] for oblivious protocols. We will see that in some cases the rigidity of oblivious protocols may cause exponential increase of their work as compared to adaptive ones.

2.4.4 Original Contribution

In the first part of this subsection we present our results on optimal work of asynchronous broadcasting against the strong adversary (i.e., the node adversary from [46]), see Table 1.

For UDG networks with known topology we get a tight result: the optimal work is $\Theta(\tau)$, where τ is the number of blocks containing at least one node. (Recall the definition of blocks, from [68], given in the previous section – here the side length for a block is $1/\sqrt{2}$.) The result holds both for adaptive and for oblivious algorithms. Our upper bound is constructive: we show an oblivious broadcasting algorithm with work $O(\tau)$. For UDG networks with unknown topology the results significantly change and they depend on whether (a lower bound on) the density d of the network is known or not. If it is known, then optimal work depends on the number τ of occupied blocks and on the granularity $g = 1/d$. We show an oblivious broadcasting algorithm with work $O(\tau\alpha^{g^2})$, for some constant $\alpha > 1$. On the other hand, we show that any broadcasting algorithm, even adaptive, must use work $\Omega(\tau\beta^{g^2})$, for some constant $\beta > 1$. If d is unknown, we show that broadcasting against the strong adversary is impossible in UDG networks.

We now summarize our results for networks modeled by graphs that need not come from configurations of points in the plane. (For such networks we assume that all nodes have distinct positive integer labels and each node knows its label.) Symmetric radio networks with known topology are those in which optimal work of asynchronous broadcasting significantly depends on the adaptivity of the algorithm. Indeed, we prove that for adaptive algorithms the optimal work is $\Theta(n)$, where n is the number of nodes in the network. The upper bound is again constructive: we show an adaptive broadcasting algorithm with work $O(n)$ working for any n-node symmetric network of known

topology. By contrast, using techniques from [46], it can be proved that any oblivious algorithm uses work $\Omega(c^n)$, for some constant $c > 1$, on some symmetric n-node network, and that there exists an oblivious algorithm working for any symmetric n-node network of known topology, using work $O(2^n)$. Hence we prove an exponential gap between optimal work required by adaptive and by oblivious broadcasting in symmetric networks of known topology. It should be noted that for arbitrary (not necessarily symmetric) networks, broadcasting with linear or even polynomial work is not always possible, even for adaptive algorithms. Indeed, it follows from [46] that exponential work (in the number n of nodes) is needed for some networks, even when the topology is known and the algorithm is adaptive. It is also shown in [46] that, for radio networks of known topology, work $O(2^n)$ is always enough.

For networks of unknown topology we have a tight result on optimal work of asynchronous broadcasting. This work is $\Theta(2^N)$, where N is the maximal label of a node, and this result does not depend on whether the networks are symmetric or not, whether the algorithm is adaptive or not, and whether the maximal label N is known to nodes or not. More precisely, we show a lower bound $\Omega(2^N)$ on the required work, even for symmetric networks with known parameter N, and even for adaptive algorithms. On the other hand, we observe that an (oblivious) algorithm described in [46] and working for arbitrary networks without using the knowledge of N has work $O(2^N)$.

In Subsection 2.4.9 we present our results on optimal work of asynchronous broadcasting against the weak adversary. Introducing this adversary was motivated by the following remark in [46]: "It would be interesting to define a weaker, but still natural, model of asynchrony in radio networks, for which polynomial-work protocols always exist." We show that if nodes are equipped with clocks, then oblivious broadcasting algorithms using work $O(n)$ for n-node networks can always be provided in the presence of the weak asynchronous adversary. This is optimal, as witnessed by the example of the line network. Local clocks at nodes need not be synchronized, we only assume that they tick at the same rate. In fact, even this assumption can be removed in most cases: our algorithm works even when the ratio of ticking rates between the fastest and the slowest clock has an upper bound known to all nodes. The exception is the case of UDG networks of unknown density (for which broadcasting against the strong adversary was proved impossible). In this special case, our algorithm against the weak adversary assumes the same ticking rate of all clocks and relies on the availability of an object obtained non-constructively: if this object is given to nodes, they can perform oblivious broadcasting with work $O(n)$.

2.4.5 Terminology and Preliminaries

A set S of positive integers is *dominated* if, for any finite subset T of S, there exists $t \in T$ such that t is larger than the sum of all $t' \neq t$ in T.

Lemma 2.4.1 *Let S be a finite dominated set and let k be its size. Then there exists $x \in S$ such that $x \geq 2^{k-1}$.*

Proof: The proof is by induction on the size k of S. If $k = 1$ then $2^0 = 1$ and the basis of induction holds.

If a set is dominated, all its subsets are dominated. By the inductive hypothesis every subset of S of size $i < k$ contains an element $x \geq 2^{i-1}$. It follows that arranging elements in S in increasing order we have $x_i \geq 2^{i-1}$, for $1 \leq i \leq k-1$. Then $\sum_{i=1}^{k-1} x_i \geq \sum_{i=1}^{k-1} 2^{i-1} = 2^{k-1} - 1$. As x_k is the largest element in S and S is dominated, we have $x_k \geq \sum_{i=1}^{k-1} x_i > 2^{k-1} - 1$, which proves the lemma. □

Any oblivious broadcasting algorithm is fully determined by the number of messages sent by each node of the network. This non-negative integer is called the *send number* of the node. For any

<table>
<tr><th></th><th>UDG networks</th><th>Symmetric Networks</th><th>Arbitrary Networks</th></tr>
<tr><td>known topology</td><td>adaptive or oblivious: $\Theta(\tau)$</td><td>adaptive: $\Theta(n)$
oblivious [46]:
$O(2^n)$
$\Omega(c^n)$, for some $c > 1$</td><td>adaptive or oblivious [46]:
$O(2^n)$
$\Omega(c^n)$, for some $c > 1$</td></tr>
<tr><td rowspan="2">unknown topology</td><td>known density d
adaptive or oblivious:
$O(\tau\alpha^{g^2})$, for some $\alpha > 1$
$\Omega(\tau\beta^{g^2})$, for some $\beta > 1$</td><td colspan="2" rowspan="2">adaptive or oblivious:
known or unknown N:
$\Theta(2^N)$</td></tr>
<tr><td>unknown density d
adaptive or oblivious:
impossible</td></tr>
</table>

Table 2.1: Optimal work of broadcasting against the strong asynchronous adversary. τ is the number of non-empty tiles, n is the number of nodes, N is the maximal label and g is the granularity of the UDG network ($g = 1/d$); c, α and β are constants.

execution of a broadcasting algorithm, a *talker* is a node that sends at least one message in this execution. Hence, for an oblivious algorithm, a talker is a node with positive send number. The following lemma is a consequence of Lemma 1 from [46].

Lemma 2.4.2 *Consider any oblivious broadcasting algorithm $\mathcal{A}$. Let u be a node in the network. Let T be the set of talkers in the in-neighborhood of u. If at least one element in T is informed by $\mathcal{A}$ and the set of send numbers of T is dominated, then u is eventually informed by $\mathcal{A}$.*

2.4.6 UDG Radio Networks

Recall the tilings of the plane defined in [68]. Let τ be the number of non-empty blocks (i.e., blocks which contain at least one node). Every 5-block contains 25 blocks, while every block contains $\Theta\left(g^2\right)$ tiles. Blocks inside a 5-block and tiles inside a block are numbered with consecutive integers (starting from 0) left to right, top to bottom. Hence every tile is assigned a pair of integers (i, j) where i is the block number in the 5-block and j is the tile number in the block. (Tiles lying in more than one block are assigned more than one such pair. This is the case when $\sqrt{2}/n \neq d$ for all n.)

As in Section 2.3, we say that two (distinct) blocks are *potentially reachable* from each other if they contain points at distance ≤ 1. Two blocks are *reachable* from each other if they contain nodes at distance ≤ 1. There are exactly 20 blocks that are potentially reachable from any given block.

Known Topology

The following algorithm is oblivious, as it consists in an assignment of send numbers to nodes.

Algorithm `UDG1`

For any pair of blocks (B, B') that are reachable from each other, Algorithm `UDG1` elects a pair of talkers (b, b') s.t. $b \in B$, $b' \in B'$, and b is at distance at most 1 from b'. Any fixed strategy (e.g., taking the smallest such pair in lexicographic order of positions) is suitable to perform the election. Notice that at most 20 talkers can be elected in every block.

Each elected talker in a 5-block is assigned a distinct label from the set $\mathcal{L} = \{0, 1, \ldots, 499\}$. This is done by partitioning the set $\mathcal{L}$ into 25 sets L_i of 20 labels each (in an arbitrary but fixed

manner). Talkers in the i-th block of any 5-block are assigned labels from set L_i. Labels in each block are assigned to talkers in increasing order according to lexicographic order of their positions.

Assignment of send numbers is done as follows: each elected talker with label i is assigned send number 2^i. If the source has not been elected, it is assigned send number 1. All other nodes are assigned send number 0. ■

Lemma 2.4.3 *Algorithm* `UDG1` *successfully performs broadcast in any UDG radio network of known topology, with work in* $O(\tau)$.

Proof: We first prove the correctness of the algorithm. As the network is connected, either $\tau = 1$ or, for any non-empty block B, there must exist a sequence of block pairs $\langle (S, X_1), (X_1, X_2), \ldots, (X_{k-1}, X_k), (X_k, B) \rangle$ such that S is the block containing the source and blocks in each pair are reachable from each other. If $\tau = 1$, all nodes in the unique non-empty block will be informed as soon as the message transmitted by the source is delivered, and algorithm `UDG1` successfully completes broadcasting with work 1. If $\tau > 1$, any non-empty block has at least one talker, and thus any node has a talker in its neighborhood. Moreover, every talker is connected to a talker located in S by a path containing only talkers.

Consider an arbitrary node v and its block B, and consider the 5-block that has B in its center (this 5-block is not necessarily part of the 5-block grid). All neighbors of v are inside this 5-block. Blocks in this 5-block are assigned distinct numbers, and thus the set of send numbers assigned to talkers in the neighborhood of v is dominated. It follows from Lemma 2.4.2 that node v will eventually receive the source message provided that at least one of the talkers in its neighborhood will receive it. Hence it is enough to show that all talkers receive the source message. This follows by induction on the length of a shortest path, in the subgraph induced by talkers, between a talker in the block S and a talker in the neighborhood of v.

In order to estimate the work of the algorithm, notice that only a constant number of nodes in each block have a positive send number, and each send number is bounded by a constant. It follows that the total work is linear in the number τ of non-empty blocks. □

Lemma 2.4.4 *The work required to complete broadcast in any UDG radio network is in* $\Omega(\tau)$.

Proof: The proof follows from the fact that at least one node in every non empty 5-block has to transmit at least once. □

Lemma 2.4.3 and Lemma 2.4.4 imply the following theorem.

Theorem 2.4.1 *The optimal work required to complete broadcast in any UDG radio network of known topology is* $\Theta(\tau)$.

Unknown Topology

When the topology of the network is unknown, elections of talkers cannot be performed without message exchanges. Here the scenario is different depending on whether (a lower bound on) the density d of the network is known or not.

The following algorithm assumes that each node is provided with the value of d. Similarly as Algorithm `UDG1` it is oblivious.

Algorithm `UDG2`

The algorithm is based on the tilings from [68] defined in the beginning of Subection 2.4.6, and works in a similar manner as Algorithm `UDG1`. The set $\mathcal{L}$ of labels is now composed of integers from the interval $\left[0, \ldots, 25 \cdot \left(\lceil \sqrt{2}/d \rceil + 1\right)^2 - 1\right]$, and it is partitioned in 25 sets L_i, each of size

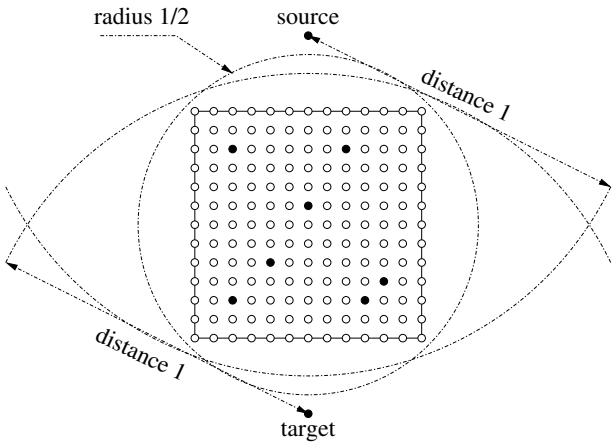

Figure 2.5: A network of the class $\mathcal{N}$ used in the proof of Theorem 2.4.2 .

$\left(\lceil\sqrt{2}/d\rceil+1\right)^2$. All nodes in the network are talkers, and each node in a 5-block gets a distinct label according to the numbering of the tile and the block it belongs to. More precisely, a node in the tile that is assigned the pair of integers (i,j) gets the label that is the jth element of L_i. Recall that there can be tiles which are partially contained in more than one block. In any case, the only node which can be contained in the tile belongs to only one block and thus its label is uniquely determined.

The send number of each node with label i is set to 2^i. ■

Proposition 2.4.1 *Algorithm* UDG2 *successfully performs broadcast in any UDG radio network of unknown topology and known density d with work in $O\left(\tau\alpha^{g^2}\right)$, for some constant $\alpha > 1$.*

Proof: The correctness of the algorithm follows from Lemma 2.4.2 by induction on the length of a shortest path from the source to an arbitrary node v.

The work of the algorithm in every block is upper bounded by $2^{\left(\lceil\sqrt{2}/d\rceil+1\right)^2}$. As $\lceil\sqrt{2}/d\rceil \in \Theta(g)$, the lemma follows. □

We now turn attention to the lower bound on the work of a broadcasting algorithm.

Theorem 2.4.2 *The work required to complete broadcast in any UDG radio network of unknown topology and known density d is in $\Omega\left(\tau\beta^{g^2}\right)$, for some constant $\beta > 1$.*

Proof: Consider the class $\mathcal{N}$ of networks depicted in Figure 2.5. The source occupies position $(0, 1.2)$ and the target occupies position $(0, 0)$. Nodes in the central part of the network are situated in an arbitrary subset of vertices of the largest regular square grid of side length d, contained in the intersection of the circles of radius 1 centered in the source and in the target, and of the circle of radius $1/2$ centered in $(0, 0.6)$. Notice that there are $\Theta\left(g^2\right)$ vertices in the grid.

The set Q of nodes situated in the grid forms a clique, and each node in Q is within distance 1 from the source and from the target. It follows that a network in $\mathcal{N}$ is connected if and only if Q is nonempty.

All nodes in Q become informed as soon as the first message sent by the source is delivered. When the first message from an informed node in Q is delivered without colliding with any delivery from other nodes in Q, broadcasting is completed successfully.

It follows that, until the completion of broadcasting, the only events that are perceived by nodes in Q are determined by deliveries of messages sent by the source. The source and the target will not receive any message until the completion of broadcasting.

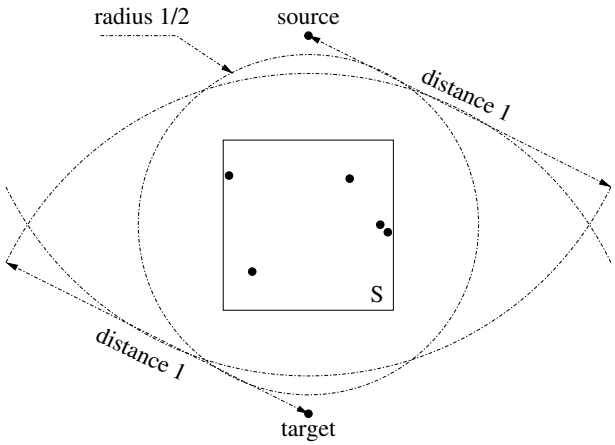

Figure 2.6: A network of the class $\mathcal{C}$ used in the proof of Theorem 2.4.3.

Consider an arbitrary adaptive algorithm $\mathcal{A}$. $\mathcal{A}$ is forced to provide a send number for the source, and it is not able to modify this number until the end of the execution (no event is perceived by the source). The adversary delays all deliveries of nodes in Q until all messages from the source have been delivered, thus guaranteeing that no node in Q can perceive an event between the first delivery of one of its messages and the end of broadcasting.

This allows us to treat $\mathcal{A}$ as an oblivious algorithm, which is obliged to provide send numbers to all nodes in the network once and forever. In fact we can assume that the algorithm assigns send numbers to vertices in the grid (a node occupying vertex p is assigned the respective send number).

Now consider a vertex p of the grid. If algorithm $\mathcal{A}$ assigns send number 0 to p, then $\mathcal{A}$ is unsuccessful in the network $N \in \mathcal{N}$ where the set Q contains only the node in vertex p. It follows that all vertices in the grid have to be assigned positive send numbers.

If the set of send numbers, assigned by $\mathcal{A}$ to vertices of the grid, is not dominated, then there exists a set T of vertices for which the largest send number x, corresponding to vertex p_0, is at most equal to the sum of all others. The adversary can make $\mathcal{A}$ unsuccessful on the network $N \in \mathcal{N}$ in which nodes in Q occupy exactly vertices from T, by letting all deliveries collide. This can be done as follows. The deliveries of messages from the node in vertex p_0 are done at times $t_1 < t_2 < \ldots < t_x$. Every other message can be delivered at one of those time points, so that at each time point t_i at least two messages are delivered.

This contradiction shows that the set of send numbers, assigned by $\mathcal{A}$ to vertices of the grid must be dominated. As the set of vertices in the grid is of size $\Theta\left(g^2\right)$ and, by Lemma 2.4.1, any dominated set on k elements contains a number $\geq 2^{k-1}$, it follows that any algorithm working correctly on all networks in $\mathcal{N}$ requires work in $\Omega\left(\beta^{g^2}\right)$, for some constant $\beta > 1$. By arranging networks of class $\mathcal{N}$ in a chain of length τ, we get a lower bound on work in $\Omega\left(\tau\beta^{g^2}\right)$ □

All results of this subsection remain valid if, instead of density d of the network, only a lower bound d' on d is known to nodes. In this case, in the formulae for the upper and lower bounds on the work, the parameter $g = 1/d$ should be replaced by $g' = 1/d'$. If nothing is known about d, however, broadcasting in UDG radio networks turns out to be impossible, as shown in the following theorem.

Theorem 2.4.3 *Broadcast in UDG radio networks of unknown topology and unknown density is impossible.*

Proof: Consider the class $\mathcal{C}$ of networks depicted in Figure 2.6. Networks in $\mathcal{C}$ are similar to

networks in class $\mathcal{N}$, defined in the proof of Theorem 2.4.2. In particular, the source and the target are located in the same positions, while the set Q of nodes is an arbitrary finite set of points in the plane, contained in the square S of side $1/2$, centered at $(0, 0.6)$. A network $C \in \mathcal{C}$ is connected if and only if Q is non empty. By following the reasoning of the proof of Lemma 2.4.2, we can show that any adaptive algorithm $\mathcal{A}$ can be treated as an oblivious one when working on a network in $\mathcal{C}$. Algorithm $\mathcal{A}$ can then be identified with a function $f : S \mapsto \mathbf{N}$ which assigns send numbers to points in the square.

First assume that the range of f is infinite and suppose that broadcasting ends with work T. This leads to a contradiction, as we can always choose a network $C \in \mathcal{C}$ with 2 nodes in Q located in two points of S that are mapped to values larger than T. By scheduling the first T deliveries of messages sent by these two nodes in the same time points, the adversary can delay completion of broadcasting until the overall work of nodes in C is at least $2T + 1$, while we assumed the total work to be exactly T.

Hence the range of f must be finite. If $f(z) = 0$, for some point $z \in S$, then broadcasting is unsuccessful on the network C in which Q contains only one node located in z. It follows that all points of S have to be mapped by f into positive integers. Then there must exist two points, x and y, such that $f(x) = f(y)$. If this is the case, the adversary can make the algorithm unsuccessful on the network C where Q contains two nodes, one in the point x and the other in the point y, by delivering messages sent by these two nodes at the same time points. □

2.4.7 Symmetric Networks of Known Topology

In symmetric networks of known topology we prove an exponential gap between the work of adaptive and oblivious algorithms. Indeed, while an adaptive algorithm can complete broadcasting on n-node symmetric networks with work in $O(n)$, an oblivious algorithm requires work in $\Omega(c^n)$, for some constant $c > 1$ (cf. [46]).

Adaptive Broadcast

The following algorithm is adaptive. Each node decides if it sends a message, after each perceived event.

Algorithm `SYM`

Knowing the topology of the network, all nodes compute the same spanning tree T, rooted at the source. Notice that, even assuming that the source is unknown to other nodes in the network, this information can be appended to the source message and thus it can be made available to each node when it is woken up by the first received message.

All internal nodes of the spanning tree T are then explored in a depth first search manner, using token-based communication in order to avoid collisions. A message is sent only after the previous message has been delivered. Algorithm `SYM` ends when the token is sent back to the source by its last internal child. ■

Lemma 2.4.5 *Algorithm* `SYM` *successfully performs broadcast in any n-node symmetric radio network of known topology with work in $O(n)$.*

Proof: We first prove correctness of Algorithm `SYM`. Since any message is sent only after the previous message has been delivered, it follows that no collision can occur during the execution of broadcasting. As all internal nodes in T transmit at least once, and T is a spanning tree of the network, all nodes will eventually receive the source message.

Since the token traverses every edge of T either 0 or 2 times, the total work of the algorithm is smaller than $2n \in O(n)$. □

As the optimal work to perform broadcasting on the n-node line is $n-1$, we have the following theorem.

Theorem 2.4.4 *The optimal work required to complete broadcasting in any n-node symmetric radio network of known topology is $\Theta(n)$.*

Oblivious Broadcast

An oblivious algorithm, performing broadcasting in *any* n-node connected radio network of known topology (not necessarily symmetric) can be obtained by arranging nodes in increasing order of labels, and assigning send number 2^{i-1} to the ith node. Such an algorithm can be proved to be correct by induction on the length of a shortest path connecting the source to an arbitrary node v, using Lemma 2.4.2. The work required to complete broadcasting by this algorithm is in $O(2^n)$.

In [46], the following network class has been introduced in order to prove that oblivious broadcasting algorithms against a more powerful adversary require work in $\Omega(c^n)$, for some constant $c > 1$.

Networks in the above mentioned class contain $\binom{k}{3} + k + 1$ nodes, for integers $k > 0$. Nodes are partitioned in three layers: the first layer contains the source, the central layer contains k nodes, while the third layer contains the remaining $\binom{k}{3}$ nodes. Edges in these networks connect the source to all nodes in the second layer, while each node in the third layer is connected to a distinct subset of 3 nodes chosen among those in the second layer. Even though edges were oriented away from the source in [46], the same proof remains valid for oblivious algorithms even if the network is made symmetric, and even against our strong adversary (which was called the node adversary in [46]).

Since the upper bound $O(2^n)$ holds for arbitrary networks and the lower bound $\Omega(c^n)$ holds even for symmetric networks, we have the following theorem.

Theorem 2.4.5 *The optimal work of an oblivious algorithm, which completes broadcasting in radio networks of known topology, is in $O(2^n)$ and in $\Omega(c^n)$, for some constant $c > 1$, both for symmetric and for arbitrary networks.*

2.4.8 Networks of Unknown Topology

For networks of unknown topology we prove matching upper and lower bounds on the optimal work of broadcasting algorithms. The upper bound we show is based on the oblivious algorithm described below, which works correctly on any network (not necessarily symmetric) containing a directed path from the source to every node. The lower bound, on the other hand, holds even on symmetric networks and for all algorithms, including the adaptive ones.

An oblivious algorithm performing broadcasting in any connected radio network of unknown topology, is obtained by assigning to node with label i send number 2^{i-1}. The algorithm works in the same manner as the one for known topology networks introduced in the previous section, but its work, instead of depending on the number of nodes of the network, depends on the largest label N appearing in the network. (N need not be known to nodes.) Thus the work of this algorithm is in $O\left(2^N\right)$. This work is proved to be optimal by the following lemma.

Lemma 2.4.6 *The work required to complete broadcasting in any symmetric radio network of unknown topology is in $\Omega(2^N)$, where N is the largest label that appears in the network.*

Proof: To prove the lemma, consider the following class $\mathcal{Z}$ of networks. Networks in the class $\mathcal{Z}$ contain a source, a target and a set R of nodes. Each node in R is connected to the source s and to the target t. The source has label 1. Nodes in $R \cup \{t\}$ are labeled with distinct integers larger

than 1, and N is the largest label appearing in $R \cup \{t\}$. R has to be non-empty, as otherwise the network would be disconnected.

The rest of the proof is based on the same idea as the proof of Lemma 2.4.2. Labels larger than 1 play the role of vertices in the grid.

As soon as a node in R delivers a message to the target without collisions, broadcasting in any network $Z \in \mathcal{Z}$ is completed. Hence, we can treat any adaptive algorithm $\mathcal{A}$ as an oblivious one, when working on networks in $\mathcal{Z}$. It follows that algorithm $\mathcal{A}$ has to assign a send number to any integer larger than 1 (which is a potential label of a node in R).

If there exists a label $\ell > 1$ such that $\mathcal{A}$ assigns send number 0 to ℓ, then $\mathcal{A}$ is unsuccessful on the network $Z \in \mathcal{Z}$ where the only node in R is labeled ℓ. It follows that $\mathcal{A}$ has to assign positive send numbers to all integers larger than 1. (Even if the maximum label N is known to $\mathcal{A}$, there is no guarantee that any particular label is assigned to a node in R, as N can be assigned to the target.) If the set of send numbers is not dominated, the adversary can make the algorithm $\mathcal{A}$ unsuccessful on the network $Z \in \mathcal{Z}$ where the (finite) set of send numbers assigned to nodes in R does not contain an element which is larger than the sum of all others (cf. the proof of Lemma 2.4.2).

As $R \cup \{t\}$ can contain up to $N - 1$ nodes, the lemma follows from Lemma 2.4.1. □

2.4.9 Broadcasting Against the Weak Adversary

In this section we present our results on the work of asynchronous broadcasting against the weak adversary. Recall that this adversary may delay delivery of messages sent by various nodes by arbitrary and unknown time intervals that may vary between nodes, but are equal for all messages sent by a given node. In this section we assume that nodes are equipped with local clocks. These clocks need not be synchronized. In one algorithm, working for UDG networks with unknown density, we assume that they tick at the same rate, and in the other, working for UDG networks with known (lower bound on) the density and also working for arbitrary networks with distinct positive integer labels, we weaken even this assumption and require only that all nodes know an upper bound on the ratio of ticking rates between the fastest and the slowest clock.

The idea of broadcasting algorithms working against the weak adversary comes from the observation that since delivery delay must be the same for all messages sent by a given node, if a node sends two messages at some time interval t, this interval may only be shifted by the adversary when delivering messages, but its length must be kept intact. Thus, using exponential intervals between just two messages sent by every node (where the exponent depends on the node label), blocking of messages can be prevented similarly as sending an exponential *number* of messages permitted preventing blocking by the strong adversary. (This is a similar work-for-time trade-off as, e.g., that in the Time-Slicing algorithm for leader election on the ring.) Due to the above possibility we can restrict the number of messages sent by every node to just 2, and thus use linear work.

We first describe an oblivious broadcasting algorithm working for networks of unknown topology whose nodes are labeled with distinct positive integers. In this algorithm we make a very weak assumption: not only clocks of nodes need not be synchronized, but they need not tick at the same rate, as long as the upper bound α on the ratio of ticking rates between the fastest and the slowest clock is known to all nodes. Without loss of generality we may assume that $\alpha \geq 2$.

Algorithm `Time-Intervals`

The source sends the message once. Upon receiving the source message, any node with label i, different from the source, sends two messages at time interval $4^{i\alpha}$ on its local clock. ■

Theorem 2.4.6 *Algorithm* `Time-Intervals` *successfully performs broadcast in an arbitrary n-node network, with work in $O(n)$.*

Proof: Since any node sends at most two messages, the work used is in $O(n)$. It remains to prove the correctness of the algorithm.

Fix the slowest ticking rate among all local clocks and call it *universal*. In the rest of the proof we will use only the universal ticking rate. Since α is the ratio of ticking rates between the fastest and the slowest clock, the (universal) time interval used by node with label i is $T_i = \frac{4^{i\alpha}}{\beta}$, where $1 \leq \beta \leq \alpha$. Fix a node u and its in-neighbors $v_1, \ldots, v_k$ that got the source message. Without loss of generality, assume that nodes v_i are ordered in increasing order of interval lengths T_i. The delivery times of messages sent by nodes v_i are $x_i, x_i + T_i$, for $i = 1, \ldots, k$. In order to prove that at least one of these messages will be heard by node u, it is enough to show that $T_k > T_1 + \cdots + T_{k-1}$. Hence it is enough to show that

$$\frac{4^{k\alpha}}{\alpha} > 4^\alpha + 4^{2\alpha} + \cdots + 4^{(k-1)\alpha}. \tag{2.3}$$

We have $\alpha^{1/\alpha} < 3$, hence

$$\frac{4^k}{\alpha^{1/\alpha}} > \frac{4}{3} \cdot 4^{k-1} > 4^1 + \cdots + 4^{k-1},$$

hence

$$\frac{4^{k\alpha}}{\alpha} > (4^1 + \cdots + 4^{k-1})^\alpha > 4^\alpha + 4^{2\alpha} + \cdots + 4^{(k-1)\alpha},$$

which proves inequality (2.3) and concludes the proof by induction on the length of the shortest path from the source to a given node. □

We now turn attention to broadcasting against the weak adversary in UDG networks. First notice that if the topology of the network is known, then Algorithm `UDG1` clearly works correctly against the weak adversary as well, and it uses the same work $O(\tau)$, which is at most $O(n)$ for n-node networks. Thus we may restrict attention to networks with unknown topology. If a lower bound on the network density is known to all nodes, then we may use the same tiling as in Algorithm `UDG2` to obtain integer labels of all nodes of the network. Subsequently we use Algorithm `Time-Intervals` and the same argument as before proves its correctness and work complexity.

The only remaining case is that of UDG radio networks in which nothing is known about the density. Recall that in this case we proved that broadcasting against the strong adversary is impossible. Somewhat surprisingly, we will show that if the adversary is weak, then broadcasting in n-node UDG networks with unknown density can be performed with work in $O(n)$. Our algorithm, however, is only of theoretical interest: its main goal is to show a situation when broadcasting is impossible against the strong adversary, but can be done using linear work against the weak adversary. The impracticality of the algorithm has two reasons. First, since it works on networks of arbitrarily small density, it requires infinite precision of the perception of Euclidean coordinates by nodes. Second, the algorithm is non-constructive: it relies on the availability of a function whose existence we prove, but which is not constructed. Once this function is given to nodes, they can perform easy broadcasting with linear work. More precisely, our algorithm relies on the following set-theoretic lemma.

Lemma 2.4.7 *There exists a function $f : \mathbf{R} \times \mathbf{R} \longrightarrow \mathbf{R}^+$ such that any distinct elements $v_1, \ldots, v_k$ and $w_1, \ldots, w_r$ from $\mathbf{R} \times \mathbf{R}$ satisfy the inequality $\pm f(v_1) \pm \cdots \pm f(v_k) \neq \pm f(w_1) \pm \cdots \pm f(w_r)$.*

Proof: Let κ be the cardinal of the continuum. Hence the cardinality of sets $\mathbf{R} \times \mathbf{R}$ and $\mathbf{R}^+$ is κ. Using the axiom of choice (this is the non-constructive ingredient in the definition of the function f), order the set $\mathbf{R} \times \mathbf{R}$ in ordinal type κ. Let $x_\gamma : \gamma < \kappa$ be this ordering. We now define the function f by transfinite induction. Suppose that $f(x_\gamma)$ is already defined, for all $\gamma < \delta$. Consider the set Z of all reals $\pm f(x_{\gamma_1}) \pm \cdots \pm f(x_{\gamma_d})$, for any finite set $\{x_{\gamma_1}, \ldots, x_{\gamma_d}\}$ of elements of $\mathbf{R} \times \mathbf{R}$,

such that $\gamma_1, \ldots, \gamma_d < \delta$. The set Z has cardinality equal to the maximum of the cardinality of δ and of $\aleph_0$ (the latter is the cardinality of the set of natural numbers). Hence the cardinality of Z is strictly less than κ, and consequently there exists a number $z \in \mathbf{R}^+ \setminus Z$. We put $f(x_\delta) = z$.

Thus the function f is defined by transfinite induction. It remains to verify that it has the desired property. Suppose by contradiction that some elements $v_1, \ldots, v_k$ and $w_1, \ldots, w_r$ from $\mathbf{R} \times \mathbf{R}$ satisfy the equality $\pm f(v_1) \pm \cdots \pm f(v_k) = \pm f(w_1) \pm \cdots \pm f(w_r)$. Let ξ be the largest index of all these elements in the ordering $x_\gamma : \gamma < \kappa$. It follows that $f(x_\xi) = \pm f(x_{\gamma_1}) \pm \cdots \pm f(x_{\gamma_d})$, for some $\gamma_1, \ldots, \gamma_d < \xi$, which contradicts the definition of $f(x_\xi)$. □

The broadcasting algorithm for UDG networks with unknown density assumes that all nodes have clocks ticking at the same rate. Given the function f whose existence follows from Lemma 2.4.7, the algorithm can be formulated as follows.

Algorithm `Non-Constructive`

The source sends the message once. Upon receiving the source message, any node with Euclidean coordinates (x, y), different from the source, sends two messages at time interval $f(x, y)$. ■

Theorem 2.4.7 *Algorithm* `Non-Constructive` *performs correct broadcasting in an arbitrary n-node UDG network, using work $O(n)$.*

Proof: As before, the complexity of the algorithm is straightforward. It remains to prove its correctness. Suppose that there exists a network with a node u that has in-neighbors $v_1, \ldots, v_k$ that got the source message. Suppose that there exist delays such that the adversary can shift time segments of lengths $f(v_1), \ldots, f(v_k)$ between messages sent by these nodes, so that all message deliveries are blocked by collisions. This implies that, for some nodes $w_1, \ldots, w_r, u_1, \ldots u_m \in \{v_1, \ldots, v_k\}$ we must have $f(w_1) + \cdots + f(w_r) = f(u_1) + \cdots + f(u_m)$, which contradicts the property of the function f established in Lemma 2.4.7. This contradiction shows that all nodes in every UDG network will eventually get the source message. □

2.4.10 Conclusion

We established upper and lower bounds on the optimal work of asynchronous broadcasting algorithms working against two types of adversaries in several classes of networks: symmetric and arbitrary directed networks and networks represented by unit disc graphs. While the complexity of most presented algorithms has been proved optimal by showing matching lower bounds, in two cases gaps between upper and lower bounds on the optimal work required by asynchronous broadcasting still remain. These gaps concern the strong adversary (bounds against the weak adversary are tight in all cases). For broadcasting in UDG radio networks of unknown topology but known density, our upper and lower bounds on optimal work are $O(\tau\alpha^{g^2})$ and $\Omega(\tau\beta^{g^2})$, respectively, for some constants $\alpha > \beta > 1$. This gap concerns both adaptive and oblivious algorithms. On the other hand, for symmetric networks of known topology, the upper and lower bounds on optimal work of oblivious algorithms are $O(2^n)$ and $\Omega(c^n)$, respectively, for some constant $1 < c < 2$. This latter gap is "inherited" from [46], where it concerned arbitrary directed networks of known topology. Closing these gaps is a natural open problem.

Chapter 3

Power Consumption

3.1 Introduction

In this chapter we consider the problem of power optimization in network communication. As in the previous chapter, the communication primitive we consider is the broadcast operation.

Minimization of the power consumption is the first goal we can think of when we consider power efficiency. We will discuss the minimum energy broadcast problem further below in this introduction.

Radio networks are often composed of small devices with batteries of limited capacity. Indeed the ability to communicate without wires is less interesting when we depend on wires to connect to a power source. In such a scenario, one of the most important aspects in power efficiency is the ability to keep the network functional for the maximum possible amount of time, as this in turn makes the network maintenance cheaper by minimizing the need for replacing/recharging the batteries.

The lifetime maximization problems vary depending on the condition defined for the network to be considered "alive" and on the connectivity condition required by the task to be performed over the network.

In the case of broadcasting, the connectivity condition requires all nodes to be reachable from the source, while other tasks like, e.g., gossiping, may impose stronger connectivity requirements. The network can be considered alive up to the time when the first node is completely drained of energy, or it can be allowed to drain a certain percentage of nodes, or even an arbitrary number of nodes, as long as the connectivity condition is respected in the residual graph composed of functional nodes (graceful degradation).

Section 3.2 considers a lifetime problem for repeated broadcast operations originating from a fixed source. The model assumed for the network topology is based on geometry and probability. Nodes are points in a given region of the Euclidean plane: the position of each node is chosen uniformly at random in that region. The condition we used to consider the network alive is that it has to be possible to find a directed spanning tree rooted at the source covering all nodes, without exceeding the residual battery charge of transmitting nodes.

Turning attention back to power minimization, we have that minimum energy broadcast on wireless networks has a strong connection with the spanning tree with many leaves problem. Spanning trees have been widely used in order to achieve power efficient broadcasting.

In wired networks (with point to point communication), the minimum overall power needed to perform broadcasting is achieved by performing it on a minimum weight spanning tree. The minimum weight spanning tree, rooted on the source of broadcasting, has to be computed on the (directed) topology graph, augmented with edge weights representing the energy needed to send a message along the edge. Thus the minimum energy broadcast problem is easily solvable in wired networks by applying well known algorithms like Chu-Liu's [51] or Edmonds's [64].

In a radio network the situation is different. Let $p(u, v)$ be the power needed by node u to send a message to node v. If u transmits with power $p(u, v)$, its message can be heard at the same time by all nodes v' such that $p(u, v') \leq p(u, v)$. It follows that, if we consider the edge set resulting from the power assigned to nodes in the directed topology graph of our radio network, all nodes pay the price of the heaviest outgoing edge only. This difference between wired and radio networks makes the minimum energy broadcast problem NP hard on radio networks.

If the model of communication is such that the reachability relation is symmetric and nodes are homogeneous and unable to adjust their transmitting power (thus they can only choose whether to transmit or be silent), the minimum energy broadcast problem requires to minimize the number of transmitting nodes and thus it can be addressed by mean of a spanning tree with many leaves. Having such a tree allows indeed to find a solution to the minimum energy broadcast problem which

is either optimal (if the source is an internal node in the spanning tree), or distant from the optimum by the cost of a single transmission (this may happen if the source is a leaf in the spanning tree and there is an alternative optimal solution of the spanning tree with many leaves problem where the source is internal).

Finding a spanning tree with many leaves is NP hard, thus it is interesting to study this problem on restricted classes of graphs, and to study variations on its definition.

In Section 3.3 we study a recently proposed variation of the spanning tree with many leaves problem, for bipartite graphs. We consider a particular subclass of graphs, i.e., regular bipartite graphs; the variation of the problem requires to maximize the number of leaves in one of the two sets composing the partition.

A more detailed definition of the problems addressed in this chapter, the related bibliographic references and a complete description of our contributions are given in the respective sections.

3.2 Random Geometric Radio Networks: Broadcast Lifetime

3.2.1 The Model and the Problem

In static ad hoc radio networks, nodes may have the ability to vary their transmitting ranges (and, thus, their energy consumption) in order to provide good network connectivity and low energy consumption at the same time. More precisely, the transmitting ranges determine a (directed) communication graph over the set V of nodes. Indeed, a node v, with range r, can transmit to another node w if and only if w belongs to the *disk* of radius r centered in v. The transmitting range of a node depends, in turn, on the energy power supplied to the node. In particular, the power P_v required by a node v to correctly transmit data to another station w must satisfy the inequality (see [142]):

$$\frac{P_v}{\mathrm{dist}(v,w)^2} \geq \eta$$

where $\mathrm{dist}(v,w)$ is the Euclidean distance between v and w, while η is a constant that, without loss of generality, can be fixed to 1.

In several previous theoretical works [5, 52, 114, 165] it is assumed that nodes can arbitrarily vary their transmitting range over the set $\{\mathrm{dist}(v,w) \mid v,w \in V\}$. However, in some network models (like sensor networks), the adopted technology allows to have only few possible transmitting range values. For this reason, we assume that nodes have the ability to choose their transmitting range from a finite set $\Gamma = \{0, r_1, r_2 \ldots, r_k\}$, $0 < r_1 < r_2 < ... < r_k$, Γ depending on the particular adopted technology (see [35,36,142]). Clearly, the maximal range value r_k in Γ must be sufficiently large to guarantee that at least one feasible solution exists. Further technical constraints on Γ will be given and discussed in Subsection 3.2.3.

A fundamental class of problems, underlying any phase of a dynamic resource allocation algorithm in ad hoc radio networks, is the one known as *range assignment problems*. Given a connectivity property Π, the objective of these problems is to find a transmitting range assignment $r : V \to \Gamma$ which induces a communication graph satisfying Π and minimizing the $\mathsf{cost}(r) = \sum r(v)^2$, i.e., the overall energy power required to deploy the assignment [114, 165].

Several research papers [5,52,165] have been devoted to the case where Π requires the transmission graph to contain a directed spanning tree rooted at a given source $s \in V$ (a broadcast tree from s). The relevance of this problem, denoted as Min Energy Broadcast, is due to the fact that any communication graph satisfying the above property allows the source to perform a broadcast operation.

Min Energy Broadcast is known to be NP hard even when $|\Gamma| = 3$ and r_1 is a small positive constant [52]. A series of constant-factor approximation algorithms are available in the literature (see, e.g., [5, 52, 72, 162]). The best known approximation factor is close to 4 and it is given in [34]. A more general version of Min Energy Broadcast is given in [30], where a not uniform *node efficiency* function $e : V \to \mathcal{R}^+$ is considered. Hence, the energy cost required to transmit from node v to w is given by $\mathrm{dist}(v,w)^2/e(v)$. This non-symmetric version of Min Energy Broadcast seems to be harder: the best known algorithm is given in [30] and yields approximation ratio $\Theta(\log n)$.

The power assignment problems do not consider some important ad hoc radio network scenarios where nodes are equipped with batteries of limited capacity and the goal is to maximize the number of broadcast operations. This important range assignment problem has been first analytically studied in [30] and it is the subject of this section.

We work in a synchronous model of communication (each node can transmit an arbitrary message in one round, nodes share a global clock). Time is divided in *time periods*. Time period t is devoted

to broadcasting the t-th message from the source s. Each node may transmit in one round only for each time period. All nodes are initially equipped with the same battery charge $B > 0$.

Definition 3.2.1 *A* range assignment schedule $\mathcal{S}$ *is a set of functions* $\{r_t : V \to \Gamma, t = 1, \ldots, m\}$.

A range assignment schedule is said to be *feasible* if, at any time period t, r_t yields a broadcast tree from s and, for any $v \in V$, it holds that

$$\sum_{t=1}^{m} \beta r_t(v)^2 \leq B.$$

The *length* of a range assignment schedule is the number of time periods. At every time period t, the battery charge of each node v is reduced by amount $\beta\, r_t(v)^2$, where $r_t(v)$ denotes the range assigned to node v during t and $\beta > 0$ is a fixed constant depending on the adopted technology. In this section, we assume $\beta = 1$, however, all our results can be easily extended to any $\beta > 0$.

The Max LifeTime problem requires to find a feasible range assignment schedule of maximal *length* m.

In [30], Max LifeTime is shown to be NP hard. In the same paper, by means of a rather involved reduction to Min Energy Broadcast with non uniform node efficiency, a polynomial time algorithm is provided, yielding approximation ratio $\Theta(\log n)$. This positive result also holds when the initial node battery charges are not uniform.

A static version of Max LifeTime has been studied in [111]: the broadcast tree is fixed during the entire schedule and the quality of solutions returned by the MST-based algorithm is investigated. Such results and techniques are not useful for solving Max LifeTime problem, as the broadcast tree may change at each time period.

Several other problems concerning network lifetime have been studied in the literature [35, 36, 111]. Their definitions vary depending on the particular node technology (i.e., fixed or adjustable node power) and on the required connectivity or covering property. However, both results and techniques (mostly of them being experimental) are not related to ours.

3.2.2 Original Contribution

To the best of our knowledge, previous analytical results on Min Energy Broadcast and Max LifeTime concern worst-case instances only. Some experimental studies on Min Energy Broadcast have been done on random geometric instances [53, 72]. Such input distributions turn out to be very important in the study of range assignment problems. On the one hand, they represent the most natural random instance family where greedy heuristics (such as the MST-based one, see [165]) have a bad behavior [72]. On the other hand, random geometric distributions provide a good model for well-spread networks located on 2-dimensional regions [35, 36, 111, 165].

We study Max LifeTime in random geometric instances of arbitrary size: the set V is formed by n nodes selected uniformly and independently at random from the 2-dimensional square of side length $\sqrt{n}$. Such instances will be simply denoted as *random sets*. Notice that the maximal Euclidean distance between two nodes in random sets is $\sqrt{2n}$, so the maximal range value r_k can be assumed to be at most $\sqrt{2n}$.

A natural and important open question is to establish whether efficiently-constructible range assignment schedules exist for Max LifeTime, having provably good length on random sets. Moreover, the design of efficient distributed implementations of such schedules is of particular relevance in ad hoc radio networks.

To this aim, as a first step we provide an upper bound on the length of an optimal range assignment schedule $\mathcal{S}$ for any finite set V in the 2-dimensional plane. Notice that this upper

bound holds for any instance, not only for random sets. When V is a random set we present an efficient centralized algorithm that, with high probability, returns a feasible schedule of length T, and T is not smaller than $1/12$ of the optimum. Here and in the sequel the term *with high probability* means that the event holds with probability at least $1 - 1/n^c$ for some constant $c > 0$.

Starting from our centralized algorithm, we design a fully distributed protocol for MAX LIFETIME. The protocol assumes that every node initially knows n and its Euclidean position only. This assumption is reasonable in static ad hoc radio networks since the position of each node can be either stored on it during deployment or it can be locally computed using a GPS system in a set-up phase. This operation is not too expensive in terms of energy consumption since it is performed only once during the set-up phase. We then show that the resulting schedule is equivalent to the one yielded by the centralized version and hence, when applied to random sets, it achieves constant approximation ratio with high probability as well. We thus get the first distributed algorithm for MAX LIFETIME having provably good performance.

We remark that in the analysis of our protocol we consider the costs related both to the construction of the range assignment schedules and to the execution of all broadcasts. Our protocol is designed to completely avoid collisions, thus no hidden cost is neglected in the analysis.

The MAX LIFETIME problem does not impose any constraint on the *completion time* of each broadcast operation, nevertheless we also provide an analysis of the *amortized* completion time of each broadcast operation produced by our protocol. If the length of the range assignment schedule generated by the protocol is T, the amortized completion time is given by the overall number of elapsed rounds divided by T.

It turns out that our protocol has amortized completion time

$$O\left(\frac{r_2 n\sqrt{n}}{T} + r_2^2 + \frac{\sqrt{n}}{r_2}\right).$$

Assume that $r_2 \in \Gamma$ is close to the *connectivity threshold* of *random geometric graphs* [59,98,146,154], i.e., $r_2 = \Theta(\sqrt{\log n})$. Then, the worst scenario for our protocol is when the initial battery charge B is very small so that T is small as well. Indeed, if $T \in O(1)$, from the previous formula we get an amortized completion time $O(n\sqrt{n \log n})$, which is a factor $\sqrt{n \log n}$ larger than the best known deterministic distributed broadcasting time in geometric radio networks without collision detection, i.e., $O(n)$ [59]. However, this protocol does not take into account *node energy* costs and, thus, the lifetime of the network. Our protocol, instead, trades completion time of each broadcast operation with global network lifetime. This fact clearly arises whenever B is large enough to allow $T \in \Omega(\sqrt{n})$ number of broadcast operations: in this case, we get $O(n\sqrt{\log n})$ amortized completion time, which is very close to the completion time of the best known distributed broadcasting algorithm.

3.2.3 Terminology and Preliminaries

A *random set* V is formed by n nodes selected uniformly and independently at random from the square Q of side length $\sqrt{n}$. The source node s can be any node in V. The length of a maximum feasible range assignment schedule (in short, schedule) for an input (V, s) is denoted as $\mathrm{opt}(V, s)$.

Given a set V of n nodes in the 2-dimensional Euclidean plane and a positive real r, the *disk graph* $G(V, r)$ is the symmetric graph where two nodes in V are linked if $\mathrm{dist}(v, w) \leq r$. When V is a random set, the resulting disk graph distribution is known as *geometric random graph* model, which is the subject of several important studies related to radio networking [59, 98, 146, 154]. In particular, it is known that, for sufficiently large n, a random geometric graph $G(V, r)$ is connected with high probability if and only if $r \geq \mu\sqrt{\log n}$, where $\mu = 1+\epsilon$ for any constant $\epsilon > 0$ [98,146,154].

The value $\mathtt{CT}(n) = \mu\sqrt{\log n}$ is known as the *connectivity threshold* of random geometric graphs.

We recall that $\Gamma = \{0, r_1, r_2 \ldots, r_k\}$ is such that $0 < r_1 < r_2 < ... < r_k \leq \sqrt{2n}$. In addition we assume that $1 \leq r_1 < \mathtt{CT}(n)$.

We define C_s to be the connected component containing s in the disk graph $G(V, r_1)$.

If $r_1 \geq \mathtt{CT}(n)$ then MAX LIFETIME on random sets admits a trivial schedule which is, with high probability, a constant factor approximation. As the source must transmit at every time period with range at least r_1 and C_s, with high probability, contains all nodes, then a feasible schedule is obtained when all other nodes are assigned the range r_1 at every time period. This motivates our assumption on r_1.

Other values in Γ can be arbitrarily fixed, provided that all of them are not smaller than $\mathtt{CT}(n)$ and at least one of them is larger than $2\sqrt{2}c\sqrt{\log n}$, where $c > \mu$ is a small constant that will be defined in Lemma 3.2.2. Informally speaking, we require that at least one value in Γ is a bit larger than the connectivity threshold. This is reasonable and relevant in energy problems related to random geometric radio networks since this value is the *minimal* one achieving global connectivity with high probability. Further discussion on such assumptions can be found in Section 3.2.7.

3.2.4 The Upper Bound

In this subsection, we provide an upper bound on the length of any feasible range assignment schedule for a set V.

Lemma 3.2.1 *Given a set V and a source $s \in V$, it holds that* $\mathrm{opt}(V, s) \leq B/r_1^2$. *Furthermore, if the size k_1 of C_s is less than n, then*

$$\mathrm{opt}(V, s) \;\leq\; \min\left\{ \frac{B}{r_1^2} \;,\; \frac{B}{r_2^4}\left(k_1 r_2^2 + r_1^2 - k_1 r_1^2\right) \right\}.$$

Proof: Since the source must transmit with range at least r_1 at any time period, the first upper bound follows easily.

If $k_1 < n$ then consider any feasible range assignment schedule $\mathcal{S}$. Let l_1 and l_2 be the number of time periods where the source transmits with range r_1 and with range *at least* r_2, respectively. It must hold that

$$l_1 r_1^2 + l_2 r_2^2 \leq B.$$

Since $k_1 < n$ then, in each of the l_1 time periods of $\mathcal{S}$, there is at least one node in $C_s - \{s\}$ having radius at least r_2. This yields

$$l_1 r_2^2 \leq (k_1 - 1)B.$$

The maximum value of $l_1 + l_2$ is achieved when $l_1 = (k_1 - 1)B/r_2^2$ and $l_2 = B/r_2^2 - (k_1 - 1)Br_1^2/r_2^4$. As $l_1 + l_2$ is the length of the schedule, we obtain the following upper bound on the number of time periods of $\mathcal{S}$.

$$l_1 + l_2 \leq \min\left\{ \frac{B}{r_1^2}, \frac{B}{r_2^4}\left(k_1 r_2^2 + r_1^2 - k_1 r_1^2\right) \right\}.$$

□

Notice that, when V is a random set, $k_1 < n$ with high probability as $r_1 < \mathtt{CT}(n)$.

3.2.5 The Algorithm

In this subsection we present a simple and efficient algorithm for MAX LIFETIME and then we analyze its performance. For the sake of simplicity, we restrict ourselves to the case $r_2 \geq 2\sqrt{2}c\sqrt{\log n}$. Nevertheless, it is easy to extend all our results to the more general assumption described in Section 3.2.3.

The following algorithm partitions Q into square cells and selects, for every time period, a set of *pivots*, i.e., nodes which are assigned range r_2. Each set of pivots is responsible for spreading the message of its time period. This message is delivered to one of the pivots from the source only using transmissions with range r_1, exploiting the subgraph C_s.

Algorithm 1 : BS (Broadcast Schedule)

Input: Set $V \subseteq Q$ of n nodes; a source $s \in V$; a battery charge $B > 0$; the range set $\Gamma = \{0, r_1, r_2 \ldots, r_k\}$.

Output: A range assignment schedule $\mathcal{S}$.

```
Partition Q into square cells of side length r_2/(2√2);
for any cell Q_j, let V_j be the set of nodes in Q_j;
construct an arbitrary ordering in V_j;
let C_s be the connected component in G(V, r_1) that contains s;
if |C_s| ≤ r_2^2 then
  W_s ← C_s;
else
  W_s ← any connected subgraph of C_s s.t. |W_s| = r_2^2 and s ∈ W_s;
end if
construct an arbitrary ordering of W_s;
for any time period t = 1, ... do
  if node with index t mod |W_s| in W_s has remaining battery charge at least r_2^2 then
    it is selected as pivot and range r_2 is assigned to it;
  else
    the algorithm stops;
  end if
  for any cell Q_j do
    if node with index t mod |V_j| in Q_j has remaining battery charge at least r_2^2 then
      it is selected as pivot and range r_2 is assigned to it;
    else
      the algorithm stops;
    end if
  end for
  all nodes in W_s not selected yet have radius r_1;
  all nodes in V \ W_s not selected yet have range 0.
end for
```

In order to analyze the performance of Algorithm BS, we will use the following lemma.

Lemma 3.2.2 *There exist two positive constants, c and γ, such that the following holds. Given a random set $V \subseteq Q$ of n nodes, and a partition of Q into square* cells *of side length ℓ, where $c\sqrt{\log n} \leq \ell \leq \sqrt{n}$, every cell contains at least $\gamma\ell^2$ nodes with high probability. The constants can be set as $c = 12$ and $\gamma = 5/6$.*

Proof: Given a fixed cell, let X_i be the random variable describing the probability for the i-th node to be inside the cell. As the ratio between the cell surface and the total surface is ℓ^2/n, $X_i = 1$ with probability ℓ^2/n and $X_i = 0$ with probability $1 - \ell^2/n$. The random variables X_i, for $1 \leq i \leq n$, are independent.

Let $X = \sum_{i=1}^{n} X_i$. The expected value of X, which describes the average number of nodes in a cell of side ℓ, is $\mathrm{E}[X] = \ell^2$.

Applying the Chernoff bound we have

$$\Pr[X < \gamma\ell^2] < e^{-\ell^2(1-\gamma)^2/2}.$$

There are at most $\lfloor\sqrt{n}/\ell\rfloor^2 \leq n/\ell^2$ cells fully contained in Q, and thus the probability of having one such cell with less than $\gamma\ell^2$ nodes can be bounded from above by $(n/\ell^2)e^{-\ell^2(1-\gamma)^2/2}$ by applying the Union Bound. As $\ell \geq c\sqrt{\log n}$, this probability is at most

$$e^{-c^2\log n(1-\gamma)^2/2+\ln\left(n/(c^2\log n)\right)} < e^{-c^2\log n(1-\gamma)^2/2+\log\left(n/(c^2\log n)\right)}.$$

Fixing $\gamma = 5/6$, if $c \geq \sqrt{144} = 12$ we have

$$\Pr[X < \gamma\ell^2] < e^{-144/72\log n+\log(n/(144\log n))} \leq e^{-2\log n+\log(n/(144\log n))} < 1/n$$

which proves the lemma. □

Theorem 3.2.1 *Let $V \subseteq Q$ be a random set of n nodes and $s \in V$ be any source node. Then, with high probability, the range assignment schedule returned by Algorithm* BS *is feasible and it has length at least β $\mathrm{opt}(V, s)$, where $\beta = 1/12$.*

Proof: Consider a fixed time period. W_s is non empty as it contains at least s. Hence it contains a pivot which is connected to s by a path only using range r_1. From Lemma 3.2.2, all cells are non empty with high probability. Hence a pivot is selected in every cell with high probability. This implies that, with high probability, the set of pivots forms a strongly-connected subgraph that covers all nodes in V. Moreover, Algorithm BS assigns, to every node, an energy power which is never larger than the current battery charge of the node. So the range assignment schedule is feasible with high probability.

We now evaluate the length T of the scheduling produced by Algorithm BS. Observe that T is the last time period of the run of Algorithm BS on input (V, s).

Let w be any node in $V \setminus W_s$; then, from Lemma 3.2.2, in its cell there are at least $\gamma\, r_2^2/8$ nodes with high probability. So, w spends at most energy

$$\left(\frac{8T}{\gamma\, r_2^2}\right) r_2^2.$$

Hence T can be any value such that

$$T \;\leq\; \frac{\gamma\, B}{8}. \tag{3.1}$$

During the schedule, every node v in W_s will have range either r_1 or r_2. Let $|W_s| = k$, then the energy spent by v is at most

$$\left(\frac{T}{k} + \frac{8T}{\gamma\, r_2^2}\right) r_2^2 + Tr_1^2. \tag{3.2}$$

Indeed in (3.2) we have to consider that a node in W_s can have range r_2 because it has been selected as pivot either of its cell or of W_s.

Now, two cases may arise:

- If $k \geq \left(\frac{r_2}{r_1}\right)^2$, since $r_1 \geq 1$, from (3.2) the amount of spent energy is at most $2Tr_1^2 + 8/\gamma \leq Tr_1^2\,(2 + 8/\gamma)$. We require B to exceed the latter value, so, T can be any value such that

$$T \leq \frac{B}{r_1^2\,(2+8/\gamma)}. \tag{3.3}$$

Observe that every value T that satisfies (3.3) also satisfies (3.1). So T can assume value $\frac{B}{r_1^2(2+8/\gamma)}$ and, from Lemma 3.2.1, we have that

$$T \geq \frac{\mathrm{opt}(V,s)}{2+8/\gamma}.$$

- If $k < \left(\frac{r_2}{r_1}\right)^2$, according to the definition of W_s, we have $k = k_1$. From (3.2) and some simple calculations, the energy spent by $v \in W_s$ is at most

$$T\,\frac{r_2^4 + k_1 r_1^2 r_2^2 + (8/\gamma)k_1 r_2^2}{r_2^2 k_1 + r_1^2 - k_1 r_1^2}$$

where we used the fact that $r_1^2 - k_1 r_1^2 \leq 0$. Observe also that, since $k_1 < \left(\frac{r_2}{r_1}\right)^2$ and $r_1 \geq 1$, we have

$$k_1 r_1^2 r_2^2 + (8/\gamma)k_1 r_2^2 \leq r_2^4\left(1 + \frac{8}{\gamma r_1^2}\right) \leq r_2^4\left(1 + \frac{8}{\gamma}\right).$$

It thus follows that the energy spent by v is at most

$$T\frac{r_2^4(2+8/\gamma)}{r_2^2 k_1 + r_1^2 - k_1 r_1^2}$$

Hence, T can be any value such that

$$T \leq \frac{r_2^2 k_1 + r_1^2 - k_1 r_1^2}{r_2^4(2+8/\gamma)}B. \tag{3.4}$$

Also in this case, every value T that satisfies (3.4) also satisfies (3.1). Finally, by combining (3.4) and Lemma 3.2.1, we get again

$$T \geq \frac{\mathrm{opt}(V,s)}{2+8/\gamma}.$$

So, the Theorem is proved for $\beta = 1/(2+8/\gamma) > 1/12$. □

We conclude this subsection observing that the time complexity of Algorithm BS is in $\Theta\left(nr_2^2 + T(n/\log n)\right)$. Indeed, partitioning nodes into cells, depending on their coordinates, and assigning an arbitrary order to nodes in each cell requires linear time. The construction of W_s can be performed by a search, and then is completed in $O(nr_2^2)$ time. Finally, for each time period t, a constant time is required to activate the right pivot for each of the $\Theta(n/\log n)$ cells. Notice that, in the interesting cases in which r_2 is close to $\mathtt{CT}(n)$, the time complexity is $O\left(n\log n + T(n/\log n)\right)$.

3.2.6 The Distributed Protocol

In this subsection, we present the distributed version of Algorithm BS. To do this, we detail the associated protocol. Our protocol is based on the same partition in cells as Algorithm BS. Cells are numbered with consecutive integers.

The aim of our protocol is to replicate the behavior of Algorithm BS in a distributed fashion. In order to do so, nodes need to acquire some knowledge of the network, with the minimum cost in terms of spent energy.

We adopt the synchronous model of node communication, i.e., the protocol acts in homogeneous *rounds*. The protocol does not require nodes to transmit spontaneously, as each node but s starts transmitting only after being woken up. We assume that every node v knows the number n of nodes, its own position (with respect to an absolute coordinate system) and, clearly, Γ.

Let $h(W_s)$ be the eccentricity of the source s in W_s, i.e. the maximum distance between s and a node in W_s. The t-th message sent by s is denoted as m_t. We assume that m_t contains the value of time period t. The described protocol is detailed in Algorithm DBS.

Our protocol has the following properties that are a key-ingredient in the performance analysis.

Fact 3.2.1 *Even though they initially do not know each other, all nodes in the same cell are activated (and disactivated) at the same round, so their local counters share the same value at every round. Furthermore, after the first $(\gamma/8)r_2^2$ broadcast operations, all nodes in the same cell know the set P of pivots of their cell and the relative order of its elements.*

Proof: A node in a cell is activated when it receives a message from the pivot of a neighboring cell, and disactivated when it receives the message sent by the pivot of its cell. As each point in a cell is within distance r_2 from all points in the same cell and in the neighboring cells, and pivots transmit with range r_2, it follows that activations and deactivations of nodes in a cell happen in the same round. This proves the first part of the claim.

The set of pivots is learnt by all nodes in a cell as a consequence of the transmissions made by the pivots. As this set is bounded to the size $(\gamma/8)r_2^2$, the proof of the claim is completed. □

In order to evaluate the length of the broadcast schedule yielded by Algorithm DBS, observe that the distributed version performs, in parallel, two tasks: (1) it constructs a broadcast communication subgraph starting from the source and (2) transmits the source message along this subgraph to all nodes. We emphasize that all node costs due to both the above tasks are taken into account: whenever a node transmits any message with range r, its battery charge is decreased by r^2.

The following lemma states the equivalence between performance of Algorithm BS and Algorithm DBS. A comparison between the transmissions made by each of the two algorithms will be used in order to prove the lemma.

Lemma 3.2.3 *Given a random set $V \subseteq Q$ and any source $s \in V$, if the length of the broadcast schedule yielded by Algorithm* BS *is T, then the length of the broadcast schedule yielded by Algorithm* DBS *is at least $T - 2$.*

Proof: Notice that the only difference in terms of power consumption between Algorithms BS and DBS lies in the Preprocessing phase required by the latter one. In that phase, at most two messages with range r_1 are sent by a node to discover W_s. Indeed, thanks to Fact 3.2.1, the if branch of the Broadcast procedure for nodes in V spends *time* instead of *power* in order to discover the set of Pivots of each cell. Hence, in the worst case, the distributed version performs two broadcasts less than the centralized algorithm. □

Corollary 3.2.1 *Let $V \subseteq Q$ be a random set of n nodes and $s \in V$ be any source node. Then, with high probability, the range assignment schedule yielded by Algorithm* DBS *is feasible and it has a length at least β $\mathrm{opt}(V, s) - 2$ where $\beta = 1/12$.*

Algorithm 2 : DBS (Distributed Broadcast Schedule)

Preprocessing: construction of $W_s \subseteq C_s$ such that $h(W_s) \leq r_2^2$

One-to-All

Starting from s, use round robin and transmitting range r_1 to inform all nodes in C_s that are at most within r_2^2 hops from s: such nodes will form W_s.

The one-to-all operation induces a spanning tree `Tree` of W_s rooted at s.

All-to-One

By a simple bottom-up process on `Tree` and using round robin on each level, s collects all node labels and the structure of `Tree`.

Initialization:

Every node sets a local counter `counter` $= -1$. Furthermore, each node has a local array P of length $(\gamma/8)r_2^2$ where it will store the ordered list of the first $(\gamma/8)r_2^2$ labels belonging to its own cell. This array is initially empty.

Let us observe that at the end of the Preprocessing phase, s has full knowledge of W_s.

Broadcast operations:

for $t = 0, 1, \ldots$ /* time periods */ **do**
  Execute Procedure BROADCAST(m_t)
end for

Procedure BROADCAST(m_t)

Nodes in W_s only:

- s selects the (t mod $\min\{|W_s|, r_2^2\}$)-th node in W_s as pivot (range r_2 will be assigned to it);

  s transmits, with range r_1, $\langle m_t, P\rangle$ where P is the path in `Tree` from s to the pivot.
- When a node in W_s receives $\langle m_t, P\rangle$, it checks whether its label is the first in P. If this is the case, it transmits, with range r_1, $\langle m_t, P'\rangle$ where P' is the residual path to the pivot.
- When the selected pivot p of W_s receives $\langle m_t, P = (p)\rangle$, it transmits, with range r_2, $\langle m_t, i\rangle$ where i is the index of its cell.

All nodes: (included the ones in W_s)

- If $(t \leq (\gamma/8)r_2^2)$ then
  - When a node v receives for the first time $\langle m_t, i\rangle$ in the time period t from the pivot of a neighbor cell i, it becomes active.
  - An active node, at every round, increments `counter` by one and checks whether its label is equal to the value of its `counter`. If this is the case, it becomes the pivot of its cell and transmits, with range r_2, $\langle m_t, i\rangle$ where i is the index of its cell.
  - When an active node in cell i receives $\langle m_t, i\rangle$, it (so the pivot as well) records in $P[t]$ the current value of counter c, i.e. the label of the pivot, and becomes inactive.
- else (i.e. $(t > (\gamma/8)r_2^2)$)
  - When a node v receives for the first time $\langle m_t, i\rangle$ in time period t from the pivot of a neighbor cell i, it checks if its label is equal to $P[t \bmod (\gamma/8)r_2^2]$. If this is the case, it becomes the pivot of its cell and transmits, with range r_2, $\langle m_t, j\rangle$ where j is the index of its cell.

Proof: By Lemma 3.2.3, if T is the length of the schedule yielded by Algorithm BS, Algorithm DBS produces a schedule of length at least $T-2$. As by Theorem 3.2.1 the lengh of the schedule yielded by Algorithm BS is, with high probability, at least 1/12 of the optimum, the proof follows. □

We now evaluate the message and the time complexity of Algorithm DBS.

Lemma 3.2.4 *The overall number of node transmissions (i.e. the message complexity) of Algorithm* DBS *is* $O(|W_s| + T \cdot ((n/r_2^2) + r_2^2))$*, where* T *is the length of the schedule.*

Proof: Observe that in the Preprocessing phase only nodes in C_s can exchange messages. In particular, s and all nodes in C_s at r_2^2 hops from s send only one message; all other nodes within 1 and $r_2^2 - 1$ hops from s send two messages. It follows that the message complexity of the Preprocessing phase is $\Theta(|W_s|)$. During each broadcast, exactly one message per cell is sent (see Fact 3.2.1). As there are $O(n/r_2^2)$ cells, in each time period $O(n/r_2^2)$ messages are exchanged. The only other messages which are sent are the ones required to reach the designated pivot in W_s, which number is bounded by r_2^2, thus proving the lemma. □

Theorem 3.2.2 *The overall number of rounds required by Algorithm* DBS *to perform* T *broadcast operations is*

$$O(r_2 n\sqrt{n} + T \cdot (r_2^2 + \sqrt{n}/r_2)).$$

Proof: For a single broadcast operation performed by Algorithm DBS, we define the *delay* of a cell as the number of rounds from its activation time to the selection of its pivot. Observe that the sum of delays introduced by a cell during the first $(\gamma/8)r_2^2$ broadcasts is at most $n - (\gamma/8)r_2^2$. Indeed, when a pivot transmits, one position in the pivots array is fixed, while each round during which there is silence allows nodes in the cell to learn that the node with the corresponding label is not in the cell. As each cell contains at least $(\gamma/8)r_2^2$ nodes with high probability, the delay introduced by each cell before completing the set is at most $n - (\gamma/8)r_2^2$. Once the set of pivots is completed (i.e., after the first $(\gamma/8)r_2^2$ broadcasts), the delay of any cell becomes 0 for all the remaining broadcasts. Moreover, a broadcast can traverse at most $O(\sqrt{n}/r_2)$ cells (as the side length of each cell is $\Theta(r_2)$, while the diameter of Q is $\Theta(\sqrt{n})$). By assuming that a maximal length path (this length being $\Theta(\sqrt{n}/r_2)$) together with maximal cell delay can be found in each of the first $\min\{(\gamma/8)r_2^2, T\}$ broadcasts, we can bound the maximal overall delay with $O(r_2 n\sqrt{n})$ rounds.

In the Preprocessing phase, Algorithm DBS uses round robin to avoid collisions. During the All-to-One phase, each node needs to collect all messages from its children before sending a message to its parent in `Tree`. Hence, the whole phase is completed in $O(nr_2^2)$ rounds, as the height of `Tree` is bounded by r_2^2.

Finally, the number of rounds required by every broadcast without delays and the preprocessing time is $O(r_2^2 + \sqrt{n}/r_2)$, since r_2^2 is the upper bound on the height of `Tree` and the length of any path on the broadcast tree outside W_s is $O(\sqrt{n}/r_2)$.

By combining the three contributions, we get the theorem bound without considering collisions among cell pivots. In order to avoid such collisions, we further organize Algorithm DBS into iterative phases: in every phase, only cells with not colliding pivot transmissions are active. Since the number of cells that can interfere with a given cell is constant, this further scheduling will increase the overall time of DBS by a constant factor only. This iterative process can be efficiently performed in a distributed way since every node knows n and its position, so it knows its cell. □

From Theorem 3.2.2, the amortized completion time of a single broadcast operation performed by Algorithm DBS is

$$O\left(\frac{r_2 n\sqrt{n}}{T} + r_2^2 + \frac{\sqrt{n}}{r_2}\right).$$

Since our protocol returns an almost maximal number T of broadcast operations with high probability, unless the available battery charge of nodes is small, the analysis we made at the end of Subsection 3.2.2 on the amortized completion time of Algorithm DBS is likely to fall in a scenario in which T is large enough to shrink the gap between our solution and the best known broadcasting time [59].

3.2.7 Conclusion

In this section, we studied the MAX LIFETIME problem on random sets. Further interesting future studies should address other basic operations as, for instance, the gossiping operation which is known to be NP hard as well [30]. A more technical problem, left open by our research, is the study of MAX LIFETIME when Γ contains more than one *positive* values *smaller* than the connectivity threshold $\mathtt{CT}(n)$ of random geometric graphs. This case seems to be very hard since it concerns the size and the structure of the connected components of such random graphs *under* the connectivity threshold [98, 146].

3.3 Spanning Trees with many Leaves in Regular Bipartite Graphs

3.3.1 The Model and the Problem

The problem of finding spanning trees with many leaves has been thoroughly investigated [16,19,63, 78,85,95,96,115,126,129–131,158]. It is known to be NP hard [63]. Lu and Ravi [130,131] provided 3-approximation algorithms and a 2-approximation algorithm was presented by Solis-Oba [158]. It is known that the problem remains NP hard even if the input is restricted to d-regular graphs (i.e., graphs whose nodes have degree d) for any fixed $d \geq 3$ [126]. A 7/4 approximation algorithm for cubic graphs is presented in [129]. Finding approximation algorithms with ratio less than 2 for d-regular graphs remains an open problem for $d \geq 4$.

The NP hardness of the optimization problem leads to seek constructive proofs for related extremal problems. A constructive proof that all graphs in a particular class have spanning trees with at least m leaves becomes an algorithm to produce such a tree for graphs in this class. Let $l(n,d)$ be the maximum integer m such that every connected n-nodes graph with minimum node degree at least d has a spanning tree with at least m leaves. The value $l(n,d)$ is known for $d \leq 5$. Trivially $l(n,2) = 2$. Storer [159] proved that $l(n,3) = \lceil n/4 + 2 \rceil$. Griggs and Wu [96] and Kleitman and West [115] proved $l(n,4) = \left\lceil \frac{2}{5}n + \frac{8}{5} \right\rceil$. In [96] it is also proved that $l(n,5) = \left\lceil \frac{3}{6}n + 2 \right\rceil$. For $d \geq 6$ the exact value of $l(n,d)$ remains unknown. For more information about this topic see [37].

In [149] a variation of the maximum leaf spanning tree problem has been introduced. This variation restricts the problem to bipartite graphs and asks to find a spanning tree having the maximum number of leaves in one of the sets composing the partition.

In [127] it is proved that this variation of the problem is NP hard for planar bipartite graphs.

In this section we study the variation of the spanning tree with many leaves problem proposed in [149], restricted to the class of regular bipartite graphs.

3.3.2 Original Contribution

We prove that the variation for bipartite graphs of the spanning tree with many leaves problem is NP hard for d-regular bipartite graphs for any fixed $d \geq 4$. We remark that our proof of NP hardness relies on a construction involving **non planar** regular bipartite graphs. It remains an open problem to determine if the problem is NP hard for regular planar bipartite graphs.

We present greedy algorithms with linear time complexity that find, for any d-regular graph, a spanning tree having a constant fraction of the maximum number of black leaves. Our algorithm for d-regular bipartite graphs provides an approximation ratio of $2 - 2/(d-1)^2$. The analysis of the performance ratio is based on the assumption that $d \geq 4$: in order to reach approximation ratio 1.5 on cubic bipartite graphs we present a refinement on our base algorithm based on local optimization.

Define $l_B(n,d)$ as the maximum m such that every d-regular bipartite graph with n black nodes has a spanning tree with at least m black leaves. Trivially $l_B(n,2) = 1$. We prove that $l_B(n,3) = \lceil n/3 \rceil + 1$. For $d \geq 4$ the exact value of $l_B(n,d)$ remains unknown, however we provide upper and lower bounds. More precisely we prove that $\left\lceil \frac{d-1}{2d}n + \frac{(d-1)^2}{2d} \right\rceil \leq l_B(n,d) \leq \left\lceil \frac{d-2}{d}n \right\rceil + 1$.

3.3.3 Terminology and Preliminaries

Let G be a graph; we use $V(G)$ to denote the set of nodes in G and $E(G)$ to denote the set of edges in G. For a node v in $V(G)$, $\Gamma_G(v)$ denotes the set of neighbors of v in G. We denote by G_d a d-regular bipartite graph. We use colors *black* and *white* to identify the two sets of the partition. Thus given a graph G_d, we seek for a spanning tree maximizing the number of black leaves. As in any regular bipartite graph the number of black nodes is equal to the number of white nodes, we

use the letter n to denote the number of black (white) nodes in G_d (clearly, the total number of nodes in G_d is equal to $2n$).

Given a spanning tree T of G_d, $\lambda_i(T)$, $1 \leq i \leq d$, denotes the number of black nodes of degree i in T. We omit T when it is clear form the context.

Lemma 3.3.1 *Let T be a spanning tree of G_d, we have*

$$\lambda_1(T) = 1 + \sum_{i=3}^{d}(i-2)\lambda_i(T). \tag{3.5}$$

Proof: As T spans all the nodes in G_d, it holds that $\sum_{i=1}^{d}\lambda_i = n$. Any edge in T connects a black node and a white node, thus the total number of edges is given by the sum of the degree of the black (or white) nodes. Hence we have that $\sum_{i=1}^{d} i\lambda_i = 2n-1$. From these two equations we obtain Equation 3.5. □

3.3.4 Regular Bipartite Graphs

We start our analysis from the general case of regular bipartite graphs, while in Subsection 3.3.5 we focus on cubic graphs to refine our results in this restricted case.

Lemma 3.3.2 *Let T be a spanning tree of G_d, then $\lambda_1(T) \leq \left\lfloor \frac{(d-2)n+1}{d-1} \right\rfloor$.*

Proof: From Equation 3.5, as $\sum_{i=1}^{d}\lambda_i = n$, we have that λ_1 is maximized when $\lambda_1 + \lambda_d = n$ and $\lambda_2, \lambda_3, \ldots, \lambda_{d-1}$ are all 0. Thus $\lambda_1 \leq \frac{(d-2)n+1}{d-1}$ and the thesis follows from the fact that we search for integer solutions. □

We now describe the algorithm `Span` that, given a graph G_d, produces a spanning tree T_A for G_d. The algorithm first builds a forest $\mathcal{F}$, then it connects the trees in $\mathcal{F}$ and the isolated nodes to form T_A. Every tree T_i in $\mathcal{F}$ is built by first choosing a black node v such that $\Gamma_{G_d}(v) \cap V(\mathcal{F}) = \emptyset$. Each tree T_i is augmented as long as a new black node w with at most $d-2$ neighbors in T_i can be found. When a tree T_i cannot be augmented, the algorithm starts building a new tree T_{i+1}.

In the following we formalize Algorithm `Span` by providing its pseudo code.

Algorithm 3 `Span`(G_d)

```
1: F ← ∅
2: i ← 1
3: while ∃ a black node v ∈ G_d \ V(F) such that Γ_{G_d}(v) ∩ V(F) = ∅ do
4:    E(T_i) ← {(v, x) such that x ∈ Γ_{G_d}(v)}
5:    V(T_i) ← {v} ∪ Γ_{G_d}(v)
6:    while ∃ a black node w ∈ G_d \ (V(F) ∪ V(T_i)) and a white node y ∈ V(T_i) such that
      |Γ_{G_d}(w) ∩ (V(F) ∪ V(T_i))| ≤ d − 2 and (w, y) ∈ E(G_d) do
7:       E(T_i) ← E(T_i) ∪ {(w, y)} ∪ {(w, z) such that z ∈ Γ_{G_d}(w) \ Γ_{T_i}(w)}
8:       V(T_i) ← V(T_i) ∪ {w} ∪ Γ_{G_d}(w)
9:    end while
10:   F ← F ∪ T_i
11:   i ← i + 1
12: end while
13: build T_A by connecting F and all the nodes in V(G_d) − V(F) in a tree
14: return T_A
```

Lemma 3.3.3 *For any G_d with $d \geq 4$ there exists a spanning tree T_A such that*

$$\lambda_1(T_A) \geq \left\lceil \frac{d-1}{2d} n + \frac{(d-1)^2}{2d} \right\rceil .$$

Proof: The proof of this lemma consists in the performance evaluation of Algorithm `Span`.

Let $T_1, T_2, \ldots, T_k$ be the trees built by the algorithm with input G_d. Any black node in a tree T_i has degree at least 3 and all the black nodes that do not belong to any tree have at least $d-1$ neighbors belonging to one tree T_h for some value $1 \leq h \leq k$. Let $\overline{b}_i$ be the number of black nodes in $V(G_d) \setminus V(\mathcal{F})$ having at least $d-1$ neighbors in T_i. Obviously,

$$n = \sum_{i=1}^{k} \left(\overline{b}_i + \sum_{j=3}^{d} \lambda_j(T_i) \right). \tag{3.6}$$

Now we bound the number $\overline{b}_i$ with $\left\lfloor \frac{d\left(1+\sum_{j=3}^{d}(j-2)\lambda_j(T_i)\right)}{d-1} \right\rfloor$ noting that T_i contains $1 + \sum_{j=3}^{d}(j-1)\lambda_j(T_i)$ white nodes and hence there are $d(1 + \sum_{j=3}^{d}(j-2)\lambda_j(T_i))$ edges from nodes in $V(T_i)$ to nodes in $V(G_d) \setminus V(T_i)$.

From Equation 3.6 we have $n \leq \frac{dk+\sum_{j=3}^{d}(dj-d-1)\lambda_j(T_A)}{d-1}$ and as in the forest each tree T_i has at least one node of degree d (i.e. the root) we have that

$$n \leq \frac{d\lambda_d(T_A) + \sum_{j=3}^{d}(dj-d-1)\lambda_j(T_A)}{d-1}. \tag{3.7}$$

If $d \geq 4$, $d^2 - 4d + 1 > 0$, and as $\lambda_d(T_A) \geq 1$ the following chain of inequalities holds:

$$\begin{aligned}
&d\lambda_d(T_A) + \sum_{j=3}^{d} (dj - d - 1)\, \lambda_j(T_A) = \\
&= d\lambda_d(T_A) + \sum_{j=3}^{d} 2d\,(j-2)\, \lambda_j(T_A) - \sum_{j=3}^{d} ((j-3)d + 1)\, \lambda_j(T_A) \leq \\
&\leq d\lambda_d(T_A) + 2d\,(\lambda_1(T_A) - 1) - ((d-3)d+1)\, \lambda_d(T_A) = \\
&= 2d\lambda_1(T_A) - 2d - \lambda_d(T_A)(d^2 - 4d + 1) \leq \\
&\leq 2d\lambda_1(T_A) - 2d - (d^2 - 4d + 1) = 2d\lambda_1(T_A) - (d-1)^2 .
\end{aligned}$$

Hence from Inequality 3.7 we have

$$n \leq \frac{2d\lambda_1(T_A) - (d-1)^2}{d-1}$$

and the thesis follows. □

Combining Lemma 3.3.3 and Lemma 3.3.2 we state the following theorem:

Theorem 3.3.1 *The problem of finding a Spanning Tree with the maximum number of black leaves for a d-regular bipartite graph G_d, $d \geq 4$, can be solved approximately by an algorithm running in linear time with approximation ratio at most $2 - 2/(d-1)^2$.*

Proof: Let T^* be a spanning tree of G_d with the maximum number of black leaves and let T_A be the spanning tree of G_d produced by Algorithm `Span`. From Lemma 3.3.2 we have that $\lambda_1(T^*) \leq \left\lfloor \frac{(d-2)n+1}{d-1} \right\rfloor$ and from Lemma 3.3.3 we have $\lambda_1(T_A) \geq \frac{d-1}{2d} n + \frac{(d-1)^2}{2d}$. It follows that $\frac{\lambda_1(T^*)}{\lambda_1(T_A)} \leq 2\left(\frac{d(d-2)n+d}{(d-1)^2 n+(d-1)^3}\right) < 2\frac{d(d-2)}{(d-1)^2} = 2 - 2/(d-1)^2$. □

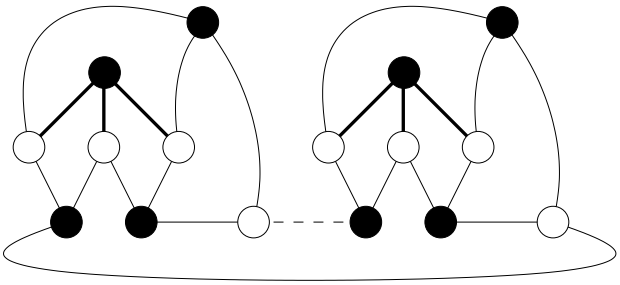

Figure 3.1: Worst case example for Algorithm `Span` on cubic graphs.

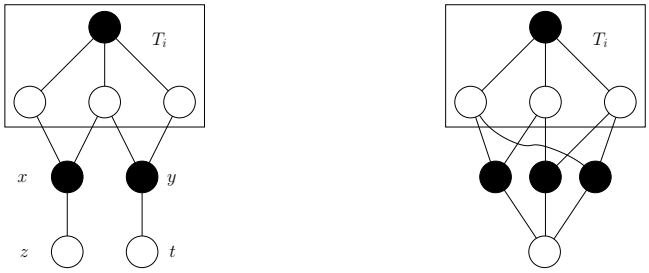

Figure 3.2: a) Pattern for local optimization in Algorithm $\texttt{Span}_3$. b) Performance analysis case.

3.3.5 Cubic Bipartite Graphs

Algorithm `Span` can obviously be applied to a graph G_3. An analysis for the case $d = 3$ gives $\lambda_1(T_A) \geq n/4$ and combining this with Lemma 3.3.2 we obtain an approximation ratio of 2.

Now consider the example in Figure 3.1: the graph in the example is a necklace composed by l repetitions of the same block. A run of Algorithm `Span` can produce a forest where every tree T_i contains a black node only (thick edges in figure represent the edges in the forest). In such a case, $\lambda_3 = l$ and so $\lambda_1 = l + 1$, while the optimum solution can assign degree 3 to all but one the black nodes on the bottom, thus achieving $2l$ black leaves.

As suggested by the example above, the performance ratio of Algorithm `Span` on cubic bipartite graphs can be improved by adding a procedure that tries to reduce the number of trees with only one black node in $\mathcal{F}$. We call $\texttt{Span}_3$ the modified version of Algorithm `Span`. If a tree T_i has only one black node at the end of the while loop of line 6 of Algorithm `Span`, $\texttt{Span}_3$ searches in $G_d \setminus \mathcal{F}$ for the pattern depicted in Figure 3.2.a. If such a pattern is present, T_i is destroyed and rebuilt starting from node x. Then, the augmenting process of lines 6 to 9 of the original algorithm is applied, resulting in a tree with at least 2 black nodes.

Lemma 3.3.4 *For any G_3 there exists a spanning tree T_A such that $\lambda_1(T_A) \geq \left\lceil \frac{n}{3} \right\rceil + 1$.*

Proof: The proof of this lemma consists in the performance evaluation of Algorithm $\texttt{Span}_3$. In the following we assume G_3 has at least 5 black nodes (if $n = 3$ or $n = 4$, the lemma trivially holds as any spanning tree having one black node of degree 3 and 2 black leaves is optimal). Let $T_1, T_2, \ldots, T_k$ be the trees built by Algorithm $\texttt{Span}_3$ with input G_3. All black nodes in $V(\mathcal{F})$ have degree 3 while black nodes in $V(G_3) \setminus V(\mathcal{F})$ have 2 neighbors in some tree $T_i \in \mathcal{F}$. Let $\overline{b}_i$ be the number of black nodes outside $\mathcal{F}$ that have at least 2 neighbors in the tree $T_i \in \mathcal{F}$. It holds that

$$n = \sum_{i=1}^{k} \left(\overline{b}_i + \lambda_3(T_i) \right).$$

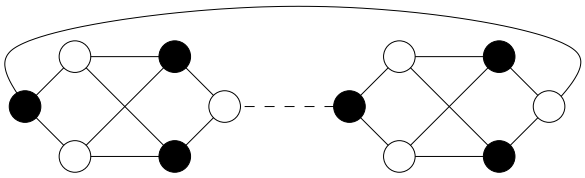

Figure 3.3: G_3^{ε}.

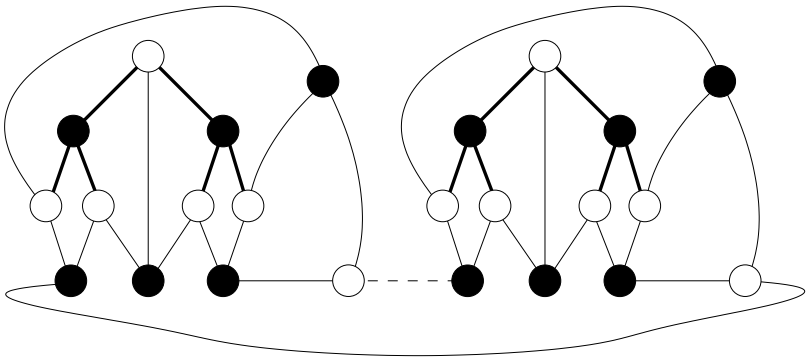

Figure 3.4: Worst case example for Algorithm $\texttt{Span}_3$.

Now we prove that $\overline{b}_i \leq 2\lambda_3(T_i)$ for all $1 \leq i \leq k$: notice that this is enough to prove the lemma since it implies that $n \leq 3\lambda_3(T_A) = 3\lambda_1(T_A) - 3$.

Using the same reasoning as in Lemma 3.3.3, we can bound $\overline{b}_i$ with $\left\lfloor \frac{3\lambda_3(T_i)+3}{2} \right\rfloor$. If $\lambda_3(T_i) \geq 2$ it holds $\overline{b}_i \leq 2\lambda_3(T_i)$ so the only case we need to analyze is when $\lambda_3(T_i) = 1$.

Assume by contradiction that $\overline{b}_1 = 3$: there must be 3 black nodes, say $x, y,$ and z, such that each of them has two white neighbors in T_i. Now consider the set W of white neighbors of $x, y,$ and z outside T_i. The condition $|W| \geq 2$ is not achievable otherwise the pattern of Figure 3.2.a would have been detected by Algorithm $\texttt{Span}_3$ and T_i would have been a tree with at least 2 black nodes. On the other hand, $|W| = 1$ is also not possible as the only G_3 where this happens is the one depicted in Figure 3.2.b: this graph has 4 black nodes only while we assumed $n \geq 5$. It follows that $\overline{b}_i \leq 2$ and the proof is completed. □

Remark 3.3.1 *The lower bound given in Lemma 3.3.4 is tight. For a given value n we can build a graph composed by $l = \lfloor n/3 \rfloor$ subgraphs closed on a necklace. The first $l-1$ subgraphs are repetitions of $k_{3,3} - e$ while the last one can have $3, 4$ or 5 black and white nodes depending on the value of $n \bmod 3$ (see Figure 3.3 for an example where $n \bmod 3 = 0$).*

Any spanning tree has to assign degree greater than 1 to at least 2 black nodes in any subgraph but one. It follows that $\lambda_1 \leq n - 2\lfloor n/3 \rfloor + 1 = \lceil n/3 \rceil + 1$.

It follows from the above remark that Algorithm $\texttt{Span}_3$ is optimal with respect to the number of guaranteed black leaves.

Lemma 3.3.2 and Lemma 3.3.4 imply the following theorem.

Theorem 3.3.2 *The problem of finding a Spanning Tree with the maximum number of black leaves for a cubic bipartite graph G_3 can be approximated by an algorithm running in linear time with approximation ratio $\leq 3/2$.*

Remark 3.3.2 *The analysis of Algorithm* $\mathtt{Span}_3$ *is tight. Consider the example in Figure 3.4: the graph in the example is a necklace composed by* l *repetitions of the same block. A run of Algorithm* $\mathtt{Span}_3$ *can produce a forest where every tree* T_i *contains two black nodes only (thick edges in figure represent the edges in the forest). In such a case,* $\lambda_3 = 2l$ *and so* $\lambda_1 = 2l + 1$*, while the optimum solution can assign degree* 3 *to all but one the black nodes on the bottom, thus achieving* $3l$ *black leaves.*

3.3.6 NP Hardness

In this subsection we prove that the problem of finding a spanning tree with the maximum number of black leaves on a 4-regular bipartite graph is NP hard. Some details of the extension of the proof for any fixed $d \geq 4$ are omitted. Our proof relies in a reduction to a restricted version of the well known NP complete problem 3-*exact cover*. 3-exact cover (in short 3EC) requires, given a universe $\mathcal{U}$ and a collection $\mathcal{S}$ of 3-subsets of $\mathcal{U}$, to determine if there exists a subcollection $\mathcal{S}'$ of pairwise disjoint sets in $\mathcal{S}$ that forms a partition of $\mathcal{U}$.

We consider instances of 3EC where each element of $\mathcal{U}$ occurs in *exactly* three subsets of $\mathcal{S}$ and $|\mathcal{U}| = 4 \cdot 3^i$, $i \geq 1$. We denote 3EC* this restricted version of 3EC.

Lemma 3.3.5 *3EC** *is NP complete.*

Proof: It is known that 3EC remains NP complete when each element of $\mathcal{U}$ occurs in *at most* 3 subsets of $\mathcal{S}$ (see, e.g., [86] pag. 222). In order to prove the lemma we show that 3EC is polynomially reducible to 3EC*: given any instance in 3EC we produce, in polynomial time, an instance in 3EC* which admits a solution if and only if the original instance admits a solution.

Let si_a be the number of sets containing the element a. The *missing inclusions* of a are given by $3 - \mathrm{si}_a$. The missing inclusions of an instance in 3EC are given by the sum of the missing inclusions of the elements appearing in the instance. Any instance in 3EC* has 0 missing inclusions.

As long as the missing inclusions in the instance are more than 2, we can reduce them as follows. Let a, b, and c be three elements having missing inclusions larger than 0. The elements need not be distinct: we can pick the same element twice if its missing inclusions are equal to 2. Introduce 3 new elements x, y, z and 4 new sets $\{x, y, z\}, \{x, y, a\}, \{x, z, b\}, \{y, z, c\}$. The new elements appear in 3 sets each. The resulting instance is in 3EC and its number of missing inclusions is reduced by 3. The satisfiability of the starting instance does not change with these additions as the set $\{x, y, z\}$ has to be selected in all possible solutions of the new instance.

If the number of missing inclusions in the instance is equal to 1, we can build 3 copies of the instance, each of them with disjoint sets of elements, thus creating an equivalent instance with 3 missing inclusions. Then we can obtain an instance with 0 missing inclusions applying the previous technique.

If the number of missing inclusions in the instance is equal to 2, again we build 3 copies of the instance on disjoint sets of elements. The resulting instance is equivalent to the original one and has 6 missing inclusions. By applying the previous technique twice we can obtain an equivalent instance with 0 missing inclusions.

Once we have an instance with 0 missing inclusions and $n \leq 3^i$ elements, we can build an equivalent one having $4 \cdot 3^i$ or $4 \cdot 3^{i+1}$ elements using techniques similar to the ones used above, thus proving the lemma. □

Theorem 3.3.3 *The problem of determining, given* G_4*, if there exists a spanning tree* T *of* G_4 *such that* $\lambda_2(T) = \lambda_3(T) = 0$ *is NP complete.*

Figure 3.5: a) Gadget *Ga1*. b) Gadget *Ga2*.

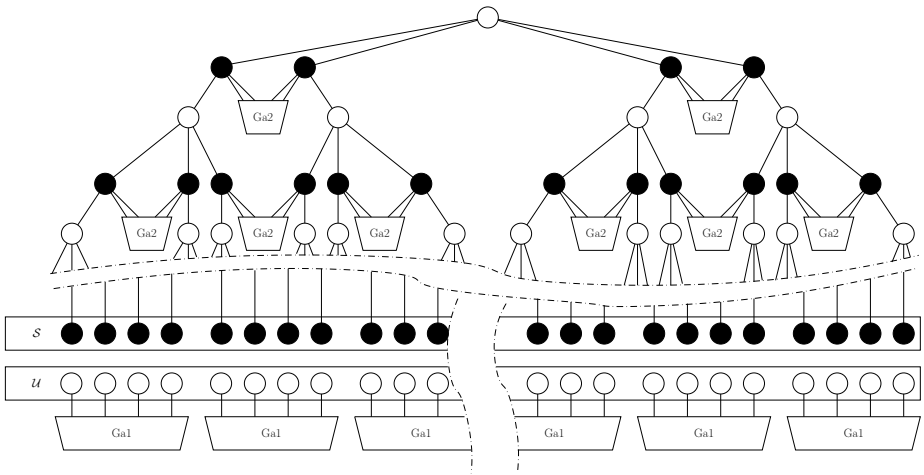

Figure 3.6: Construction of the graph used for the reduction. Edges between black nodes representing sets in $\mathcal{S}$ and white nodes representing elements of $\mathcal{U}$ are omitted.

Proof: Starting from any instance I of 3EC*, we construct in polynomial time a graph G_4, and the thesis follows from the fact that G_4 admits a spanning tree T with $\lambda_2(T) = \lambda_3(T) = 0$ if and only if I admits a solution.

To build G_4, we create a black node for each set in $\mathcal{S}$. We add new nodes to form a tree whose leaves are the $4 \cdot 3^i$ black nodes representing sets in $\mathcal{S}$. The tree is made by white nodes of degree 4 and internal black nodes of degree 2. More precisely the tree is rooted in a white node; even levels contain white nodes while odd levels contain black nodes. By construction, the tree will have $2i+1$ levels and at level $2j+1$, $j \geq 0$ there will be $4 \cdot 3^j$ black nodes (i.e., the levels with black nodes have an even number of nodes). Then we create a white node for each element of the universe $\mathcal{U}$ and we put an edge between the white node representing element $u_i \in \mathcal{U}$ and the black leaf representing set $S_j \in \mathcal{S}$ if $u_i \in S_j$.

To complete the construction, we connect each of the white nodes representing elements in $\mathcal{U}$ to the outgoing edges of a gadget Ga1 (see Figure 3.5.a) and we connect each of the black nodes with degree 2 with a gadget Ga2 twice (see Figure 3.5.b): this operation can always be done as the number of internal black nodes in the tree is even. The resulting graph is outlined in Figure 3.6.

Given a solution for I, it is easy to derive a spanning tree for G_4, having black nodes of degree 4 and 1 only, by assigning degree 4 to all the black nodes connected to gadgets Ga2 and to the black nodes representing sets forming the solution for I. On the other hand, if a spanning tree T^* of G_4 exists having black nodes of degree 4 and 1 only, a solution for I can always be found. First notice that in any such T^*, all the black nodes connected to gadgets Ga2 must have degree 4, as nodes inside the gadgets must be connected to the tree and black nodes inside the gadgets cannot be assigned degree 4 without resulting in some black node of degree 2 or 3. As a result, all the black nodes representing sets of $\mathcal{S}$ in G_4 that have degree 4 in T^* are connected from above, thus they

cannot be adjacent in T^* to the same white node representing an element of $\mathcal{U}$ without creating a loop (i.e., if we consider the sets represented by black nodes of degree 4 in T^*, they are pairwise disjoints). Finally, all the white nodes representing elements in $\mathcal{U}$ need to be connected from above in T^*, as connecting a white node from below means passing trough a gadget Ga1, thus producing at least a black node of degree 2 or two of degree 3. It follows that starting from T^*, a solution for I can be found by taking all sets of S corresponding to black nodes of degree 4 in T^*. □

Corollary 3.3.1 *The problem of finding a spanning tree with the maximum number of black leaves is NP hard for 4-regular bipartite graphs.*

Proof: Define the BLST problem as follows: given G_d and an integer k, determine whether it exists a spanning tree T of G_d such that $\lambda_1(T) \geq k$. It is easy to show that BLST on 4-regular bipartite graphs is NP complete. Indeed, from Theorem 3.3.3, we know that it is NP complete to determine if a given G_4 has a spanning tree T such that $\lambda_2(T) = \lambda_3(T) = 0$. It follows that the BLST problem is NP complete when $k = \frac{(d-2)n+1}{d-1}$. Having an algorithm that resolves in polynomial time the problem of finding a spanning tree of a given G_4 with the maximum number of black leaves, would allow us to solve BLST in polynomial time, thus proving the problem to be NP hard. □

All the proofs in this subsection can be extended to any $d \geq 5$. Generalizations of gadgets Ga1 and Ga2 for any value of d are straightforward. For even values of d, each of the $d(d-1)^j$ internal black nodes at level $2j+1$ in the graph G_d (corresponding to the graph G_4 previously defined) have $d-2$ edges going toward generalized gadgets Ga2 (thus $(d-1)^j(d-2)$ gadgets are needed at level $2j+1$), respecting the condition that each black node is connected to a gadget at least twice. The instance of $(d-1)EC^*$ contains $d(d-1)^i$ elements (and thus white nodes corresponding to elements are connected to $(d-1)^i$ generalized gadgets Ga1).

For d odd, the construction of G_d differs slightly. The tree used in the construction is rooted on a black node of degree d. Internal black nodes have degree $\lfloor d/2 \rfloor$ or $\lceil d/2 \rceil$, while internal white nodes have degree d. Every pair of internal black nodes of the tree, on level $2j$ ($j \geq 1$) is connected to a generalized gadget Ga2: one node $\lceil d/2 \rceil$ times, and the other $\lfloor d/2 \rfloor$ times (notice that as $d \geq 5$, $\lfloor d/2 \rfloor \geq 2$). Level 1 contains d white nodes, thus level 2 contains $d(d-1)$ black nodes, which is an even number. In general, level $2j+1$ contains $((d-2)/2)^j d(d-1)^j$ white nodes, while level $2(j+1)$ contains $((d-2)/2)^j d(d-1)^{j+1}$ black nodes. It follows that all even levels larger than 0 contain an even number of black nodes. Moreover, the number of black nodes in each of these levels is divisible by d.

The $((d-2)/2)^{i-1}d(d-1)^i$ black leaves on level $2i$ represent sets of the instance $\mathcal{S}$ of $(d-1)EC^*$. The same amount of white nodes, each of them connected once to one of the $((d-2)/2)^{i-1}(d-1)^i$ generalized gadget Ga1, represent elements of $\mathcal{S}$.

Hence we have the following theorem.

Theorem 3.3.4 *The problem of finding a spanning tree with the maximum number of black leaves is NP hard for d-regular bipartite graphs for any fixed $d \geq 4$.*

3.3.7 Conclusion

Spanning trees of connected graphs are a major topic of research in the area of graph algorithms. In this section we studied the problem of finding a spanning tree with the maximum number of black leaves in regular bipartite graphs.

We proved that the problem is NP hard for any fixed $d \geq 4$ and we presented a linear time algorithm that achieve approximation ratio $2 - 2/(d-1)^2$.

It is an interesting question whether this problem is NP hard or polynomial time solvable in the case of cubic bipartite graphs. Our contribution for this class of graphs is a linear time algorithm with approximation ratio 1.5.

Our proof of NP hardness relies on a construction involving regular bipartite graphs that are non planar. It remains an open problem to determine if the problem remains NP hard for regular planar graphs also. In [127] it is shown that the problem is NP hard for planar **non regular** bipartite graphs.

Finally, define $l_B(n,d)$ to be the maximum m such that every G_d with n black nodes has a spanning tree with at least m black leaves. Obviously $l_B(n,2) = 1$. From Lemma 3.3.4 and Remark 3.3.1 we have $l(n,3) = \lceil \frac{n}{3} \rceil + 1$. It would be interesting to determine precisely the value $l_B(n,d)$ for any $d \geq 4$. We know that

$$\left\lceil \frac{d-1}{2d}n + \frac{(d-1)^2}{2d} \right\rceil \leq l_B(n,d) \leq \left\lceil \frac{d-2}{d}n \right\rceil + 1$$

where the lower bound follows from Lemma 3.3.4 and the upper bound can be obtained by generalizing Remark 3.3.1, using as a building block for the necklace the graph $k_{d,d} - e$.

Chapter 4

Other Communication Tasks

4.1 Introduction

This chapter is devoted to two different problems concerning communication over networks.

The first problem, addressed in Section 4.2, is a radio-frequency assignment problem; in Section 4.3 we consider acknowledged broadcasting. While a radio-frequency assignment can be used as a building block to simplify the development of any communication protocol over a radio network, acknowledged broadcast is a specific task, which requires nodes in the network to be aware of the completion of the dissemination of the source message, i.e., the completion of the broadcast operation. This introduction is thus divided in two subsections, each of which provides an overview of the respective task.

Radio-frequency Assignment

The communication task we address in Section 4.2 is a radio-frequency assignment problem. Radio-frequency assignment problems require to find frequency assignments to nodes in the network in a way such that communication among nodes can be performed concurrently without the problem of collisions. In order to do so, the available radio spectrum is divided in non overlapping *channels*. This subdivision of the available radio spectrum however reduces the bandwidth available for each channel and thus it reduces the speed of communication. It follows that it is important to use as few channels as possible.

The formulation of the problem as a graph coloring problem is natural, but it requires to carefully consider the constraints such a coloring has to comply to.

Forbidding the assignment of the same color to nodes at distance 2 is motivated by the problem of hidden collisions, which could arise on common neighbors of nodes sharing the same color and thus operating on the same radio-frequency. This problem is called $L(1,1)$-labeling problem but is also known as the distance-two coloring of a graph or coloring of the square of the graph, and has been well-studied [2, 128, 136]. In general, in order to impose the assignment of distinct colors to nodes up to distance d from each other, we could work on the dth power of the topology graph without modifying the usual node coloring constraint of assigning distinct colors to neighbors.

Adjacent channels are more likely to cause interferences in communication due to imprecision in modulation of the transmitters. Differently from the usual definition of node coloring which does not consider any distance metric among colors, the radio-frequency assignment problem thus requires to define such a metric among frequencies. This makes it reasonable to define new coloring constraints which depend on the distance between nodes, allowing to require the assignment of distant frequencies to close nodes.

The first coloring problem with distance constraints defined in the literature is the one known as λ-coloring, proposed by Roberts and first studied by Griggs and Yeah [97]. A λ-coloring is a $L(2,1)$-labeling, i.e., it imposes to assign radio-frequencies which are 2 apart to neighboring nodes and to assign different radio-frequencies to nodes at distance 2. This definition has been generalized to the $L(h,k)$-labeling problem, and it has been widely studied in the literature (see, e.g., [23, 166] for a comprehensive survey).

A further natural generalization of the $L(h,k)$-labeling problem is the $L(\delta_1,\ldots,\delta_\sigma)$-labeling problem. This problem requires to assign colors δ_i apart to nodes at distance $i \leq \sigma$. Notice that, if $\delta_i < \delta_j$ for some $i < j$, we should use as a condition the existence of a path of a given length in order to define the constraints, as the shortest path between two nodes could lead to a weaker constraint than a longer one.

For what concerns the NP hardness, we have that the $L(2,1)$-labeling problem is NP hard even for graphs of diameter 2, planar graphs, bipartite graphs, split graphs, and chordal graphs (see [23] for references and a wider discussion). The more general $L(\delta_1,\ldots,\delta_\sigma)$-labeling problem does not

seem easier but, because of its generality, NP hardness proofs valid for all values of σ and δ_is seem rather difficult to achieve. The $L(1_1, \ldots, 1_\sigma)$-labeling problem, i.e., the distance-σ labeling problem has been proved to be NP complete by McCormic [136] for all $\sigma \geq 1$.

Our contribution addresses *outerplanar graphs*. Outerplanar graphs are planar graphs which admit an embedding in the 2-dimensional plane where all nodes lie on the external (infinite) face. Similarly to planar graphs, outerplanar graphs have a characterization by minors. A graph is outerplanar if and only if it does not contain K_4 (i.e. the complete graph on 4 nodes) or $K_{2,3}$ (i.e., the complete bipartite graph with partitions with 2 and 3 nodes) as minors.

An important characteristic of outerplanar graphs is that they are graphs of bounded treewidth. The notions of treewidth has been introduced by Robertson and Seymour [151]. In the following we report the definition of treewidth used by Bodlaender in [17]. This definition is based on *tree-decompositions*, thus the definition of tree-decomposition is also reported.

> "A tree-decomposition of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a family of subsets of V, one for each node of T, and T a tree such that:
>
> - $\cup_{i \in I} X_i = V$,
> - for all edges $(v, w) \in E$, there exists an $i \in I$ with $\{v, w\} \in X_i$,
> - for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.
>
> The treewidth of a tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G is the minimum treewidth over all possible tree-decompositions of G."

The contribution given by Bodlaender in [17] is a linear time algorithm to find tree decompositions for graphs of constant treewidth. (The decision problem "is the treewidth of a given graph at most k?" is NP complete if k is not a constant [7].)

A graph of treewidth 1 is a tree, thus the treewidth somehow measures the "distance" of the given graph from being a tree. An equivalent formulation of the treewidth of a graph can be obtained using k-trees. If a graph has treewidth $\leq k$, then there exists a k-tree containing it as a subgraph. The treewidth of outerplanar graphs is at most 2. The concept of treewidth is interesting as it plays an important role on graph minors results from Robertson and Seymour (see, e.g., [151–153]) and because many well known NP complete problems admit a polynomial (in some cases even linear) time solution on graphs of bounded treewidth given together with a tree-decomposition (see [17]). This kind of solutions however, apart from the theoretical interest, are far from being applicable in practice, as hidden constant are usually huge. This is the case of distance-σ labeling of graphs of bounded treewidth (for constant σ) [169]. The availability of an (impractical) polynomial time algorithm, however, is promising for the possibility to achieve a good level of approximation with an efficient one.

Acknowledged Broadcast

Section 4.3 considers acknowledged broadcasting.

There are two alternative definition in the literature for acknowledged broadcasting; the weaker one requires the source only to be aware of the completion of the broadcast. The stronger one requires this awareness in all nodes composing the network. Having an acknowledged broadcasting algorithm for the weaker definition of the task, however, allows us to easily obtain an acknowledged broadcasting algorithm for the stronger definition as well. The completion time of the derived algorithm would be twice the time required by the original one, thus the time complexity of the two tasks is the same.

Informing the source that broadcasting has been completed is important, e.g., when the source has to send several messages and no node should learn a given message before all nodes learned the previous one. The strong version of acknowledged broadcasting is essential, e.g., when the source message is an order to accomplish some simultaneous action of all nodes (the *Firing Squad Problem*, cf. [133]).

If the size of the network is unknown, feasibility of acknowledged broadcasting requires the network to be strongly connected. (For networks of known size, acknowledged broadcasting could be achieved even in non strongly connected networks, using time-out.)

Working in a spontaneous wake up model, Chlebus et al. [43] proved that acknowledged broadcasting is impossible, if collision detection is not available, and gave an algorithm using collision detection that works in time $O(nD)$, where n is the number of nodes and D is the eccentricity of the source. Uchida et al. [161] showed an acknowledged broadcasting algorithm without collision detection working in time $O(n^{4/3} \log^{10/3} n)$ for all strongly connected networks of size at least 2. In particular, it follows that the impossibility result from [43] is really caused by the singleton network for which acknowledged broadcasting amounts to realize that the source is alone.

Acknowledged broadcasting algorithms presented in the above mentioned papers have time complexities which are quite distant from the ones of the best known broadcasting algorithms without acknowledgement, i.e., $O(n \log^2 D)$ from [56], and $O(n \log n \log \log n)$ from [57]. This gap is not justified by a matching lower bound, thus it is interesting to study whether more efficient acknowledged broadcasting algorithms can be developed.

4.2 $L(h,1,1)$-Labeling of Outerplanar Graphs

4.2.1 The Model and the Problem

In this section we focus on $L(h,1,1)$-labeling of outerplanar graphs. An *outerplanar* graph is a graph that has a planar embedding such that all the nodes lie on the external face. A formal definition of an $L(h,1,1)$-labeling is given below.

Definition 4.2.1 *Let G be a graph and $h \geq 1$ be a non-negative integer. An $L(h,1,1)$-labeling of G is an assignment of colors (integers) to the nodes of G from the set of integers $\{0,\cdots,\lambda\}$ such that nodes of distance 1 have colors that differ by at least h, and nodes of distance 2 or 3 have colors that differ by at least 1.*

The aim of the $L(h,1,1)$-labeling problem is to minimize λ, denoted by $\lambda_{h,1,1}$ and called the *span* of the $L(h,1,1)$-labeling. The minimum number of colors used by the labeling is denoted by $\chi_{h,1,1} = \lambda_{h,1,1} + 1$. This definition of used colors is motivated by the fact that, when $h > 1$ some of the colors in $\{0,\cdots,\lambda\}$ could not be assigned to any node even in an optimal $L(h,1,1)$-labeling. Using the set $\{1,\cdots,\lambda\}$ instead of $\{0,\cdots,\lambda\}$, the span and number of used color would have been exactly equal, but all the literature on the argument adopts this notation.

We start our study by addressing the $L(1,1,1)$-labeling problem, i.e., the distance three coloring, where colors are distinct for nodes that are within distance three of each other, then we move to $L(h,1,1)$-labelings, for any $h \geq 2$.

The distance-σ coloring problem, where all nodes within distance $\sigma \geq 1$ must have distinct colors, have been studied in the literature. This problem contains as a subcase the $L(1,1,1)$-labeling problem. In [169], Kanari et al. gave an $O(n^3)$ time algorithm to $L(1^\sigma)$-label a graph with n nodes of bounded treewidth k. Outerplanar graphs are graphs of treewidth at most 2, thus the algorithm from Kanari et al. can be used to achieve an optimum coloring of any outerplanar graph with n nodes in time $O(n^3)$. However, the multiplicative constant of the algorithm (which depends on the treewidth) is already too big on graphs of treewidth 2. Indeed, for graphs of treewidth 2, the multiplicative constant is $\alpha^{2^{31}}$, where α is the chromatic number of the third power of the graph to be colored.

For outerplanar graphs, the $L(h,1)$-labeling problem for $h \geq 1$ has also been studied. The $L(1,1)$-labeling problem appeared in [18,29] and the $L(2,1)$-labeling in [18,21,29,109]. To the best of our knowledge, nothing is known for the $L(h,1,1)$-labeling of outerplanar graphs for $h \geq 2$.

4.2.2 Original Contribution

For an outerplanar graph G of maximum degree Δ we present lower bounds of $3\Delta - 3$ for the maximum number of colors that are needed to perform the $L(1,1,1)$-labeling. We show that by using a simple greedy first fit approach, $4\Delta - 7$ colors are necessary to $L(1,1,1)$-label an outerplanar graph with maximum degree Δ. Then we give a linear time approximation algorithm to $L(1,1,1)$-label any outerplanar graph of maximum degree $\Delta \geq 6$, using no more than $3\Delta + 9$ colors. Thus our lower bound on the number of colors needed and the number of colors used by the $L(1,1,1)$-labelings defined by our algorithm are within a small additive constant.

Finally, we extend our algorithm in order to $L(h,1,1)$-label an outerplanar graph using no more than $3\Delta + 2h + 7$ colors, for $\Delta \geq 4h + 7$, $h \geq 2$. Our lower bound can be applied to $L(h,1,1)$-labelings as well, but as h appears in the number of colors used by our $L(h,1,1)$-labeling algorithm, the additive constant separating our lower bound and the upper bound provided by the algorithm depends on h.

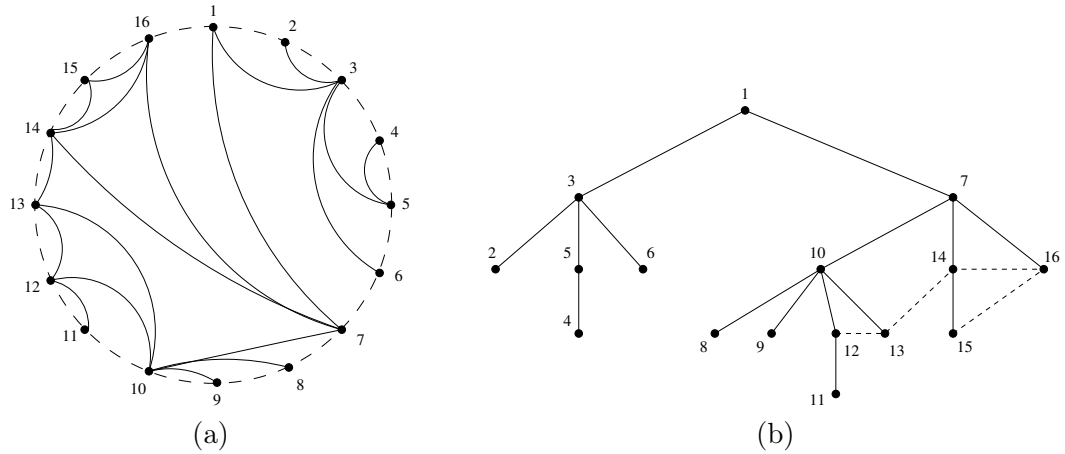

Figure 4.1: a) An Outerplanar embedding of G. b) The resulting $OBFT(G)$.

4.2.3 Terminology and Preliminaries

Let $G = (V, E)$ be a graph with node set V and edge set E. The number of nodes of the graph is denoted by n and the maximum node degree by Δ. Throughout the section we assume our graph connected, loopless and undirected.

We first state some known facts about outerplanar graphs, of which the first two are well-known.

Characterization by Minors: A graph G is outerplanar iff it does not contain the complete graph K_4 nor the complete bipartite graph $K_{2,3}$ as minors. (A *minor* of a graph is obtained by edge contractions, edge deletions or deleting isolated nodes.)

Degree 1 **or** 2: An outerplanar graph G has a node of degree 1 or 2.

OBFT(G) [29]: An outerplanar graph G has an *ordered breadth first tree graph OBFT(G)*, constructed in the following manner. Choose a node r and induce a *total ordering* on the nodes clockwise on the exterior face of a planar embedding of G. Perform a breadth first search starting with the root r and visit the nodes in order of the given ordering. We end up with an OBFT(G) with possibly some non-tree edges which have the following properties. Denoting as $v_{l,i}$ the i^{th} node from the left at level l, a non-tree edge can only exist between nodes x and y if:

1. x and y are adjacent nodes on the same level, i.e. $x = v_{l,i}$ and $y = v_{l,i+1}$ for some level $l \geq 1$ and $i \geq 1$;

2. x and y are nodes on adjacent levels, $x = v_{l,i}$ and $y = v_{l+1,j}$, and y must be the rightmost child of its parent $w = v_{l,k}$ and $k = i - 1$, i.e. node x must be the next node after w on the same level in the OBFT(G).

See Figure 4.1 for an example of OBFT(G), where dotted lines denote non-tree edges.

Given as input an outerplanar embedding of G, an OBFT(G) can be computed in $O(n)$ time.

We prove the following results concerning an OBFT(G) that will be useful to prove the upper bound on performance of our algorithms.

Lemma 4.2.1 *Let G be an outerplanar graph with its associated OBFT(G), and two siblings x and y, $x < y$, in OBFT(G). Any node u in the subtree of OBFT(G) rooted at x is less than any node w in the subtree of OBFT(G) rooted at y.*

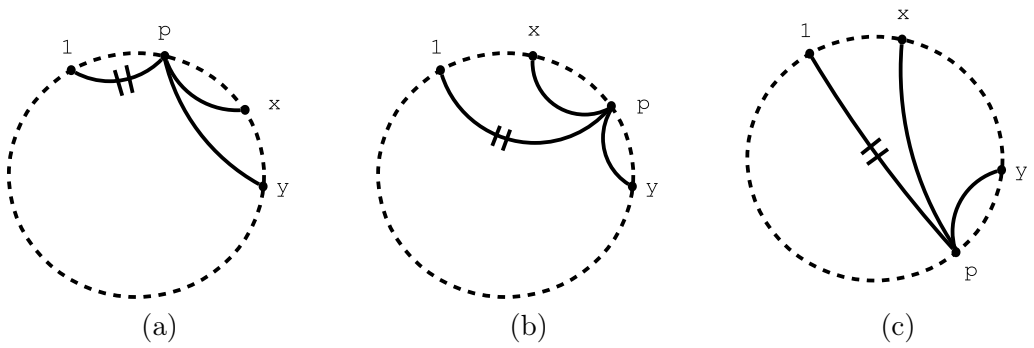

Figure 4.2: Proof of Lemma 4.2.1. Lines with double bars stand for paths while simple lines represent edges.

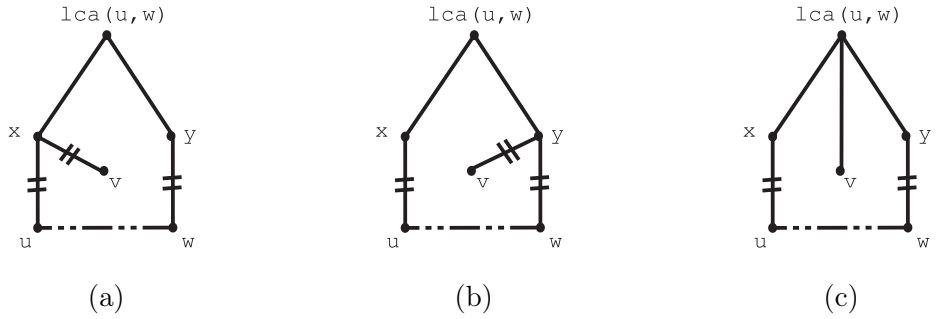

Figure 4.3: Proof of Theorem 4.2.1. Lines with double bars stand for paths while simple lines represent edges.

Proof: First observe that the parent of x and y, say p, can assume three possible relative positions with respect to x and y: $p < x < y$, $x < p < y$ and $x < y < p$ (see Figure 4.2).

In the first case (Figure 4.2.a), node u can lie either between p and x or between x and y, otherwise a crossing would be generated. Assume by contradiction that there exists a node $w < u$. Now, w cannot lie between root 1 and p (path $w \rightsquigarrow y$ would cross path $1 \rightsquigarrow p$); w cannot lie between p and x (path $w \rightsquigarrow y$ would cross edge (p, x)); so the only feasible interval for w is between x and y. Nevertheless, also in this interval, $w < u$ implies a crossing between paths $x \rightsquigarrow u$ and $y \rightsquigarrow w$. So $u < w$.

In the second case (Figure 4.2.b) $1 < u < p$ as there is necessarily a path connecting root 1 to p, and $w > p$ for similar reasons. So $u < w$.

Finally, in the third case (Figure 4.2.c) u is either between root 1 and x or between x and y. With similar reasoning as in the first case, $u < w$. □

Theorem 4.2.1 *Any OBFT(G) of an outerplanar graph G is an outerplanar embedding of G.*

Proof: First observe that, in view of the definition of outerplanar graph, if the embedding is not outerplanar, then either there exists some node embedded inside an internal face, or there is some node on the boundary of internal faces only.

Given an OBFT(G), let us suppose first that there is a node v embedded inside an internal face f. In fact, if a whole subtree is embedded inside f then we can contract it to its root, say v. We will prove the claim by contradiction. The boundary of f is the cycle created in the OBFT(G) by

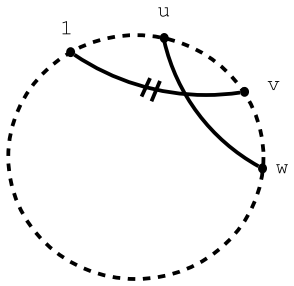

Figure 4.4: Proof of Theorem 4.2.1. Lines with double bars stand for paths while simple lines represent edges.

at least one non-tree edge (u, w) (see Figure 4.3). Let us consider the lower common ancestor of u and w on the boundary of f, and call it $lca(u, w)$. Since v is embedded inside f then $lca(u, w) \neq v$. Let x and y be the two children of $lca(u, w)$ on the boundary of f. By the OBFT(G) construction, it must be $x < y$. In view of the properties of the non-tree edges of an OBFT(G), for v to be inside f one of the following three configurations must occur:

a) v is in the subtree rooted at x (see Figure 4.3.a);

b) v is in the subtree rooted at y (see Figure 4.3.b);

c) v is a child of $lca(u, w)$ (see Figure 4.3.c).

Since in all three cases we have $u < v < w$ by Lemma 4.2.1, we get a contradiction as it is impossible to place in the outerplanar embedding of G the tree-path $1 \rightsquigarrow v$ not crossing edge (u, w) as shown in Figure 4.4. It follows that v does not exist.

Let us suppose now that a node v lies on the boundary of internal faces only and consider the simple cycle C constituted by the boundary of the union of all such faces. By construction, if v lies on level l of the OBFT(G), then on C there must be a node w on a level strictly greater than l and a node u on a level strictly less than l such that there exist paths $w \rightsquigarrow v$ and $u \rightsquigarrow v$ not using nodes of C. As u and w both lie on C, then there are two distinct paths inside C connecting u and w both passing through a node at level l. This leads to an absurdity as we can construct the forbidden minor $K_{2,3}$: v represents the internal node, u and w are the degree 3 nodes and the two nodes on level l are the remaining degree 2 nodes. □

Corollary 4.2.1 *In an OBFT(G) of an outerplanar graph G, for each node c, there exists at least one of c's children not having non-tree edges on both sides.*

Proof: The claim trivially holds if c is the root of the OBFT(G), as the rightmost child of c cannot have non tree edges on its right. In general the claim directly follows from Theorem 4.2.1 as node c would be internal (see Figure 4.5). □

4.2.4 L(1,1,1)-Labeling

In this subsection we deal with the $L(1, 1, 1)$-labeling of outerplanar graphs. The technique used here will be generalized in the next subsection in order to handle the $L(h, 1, 1)$-labelings for $h \geq 2$.

Let us begin by providing a lower bound on the number of colors needed.

Theorem 4.2.2 *There exists an outerplanar graph of degree Δ that requires at least $3\Delta - 3$ colors to be $L(1, 1, 1)$-labeled.*

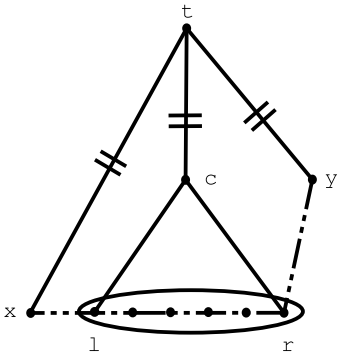

Figure 4.5: Proof of Corollary 4.2.1. Lines with double bars are paths while simple lines represent edges.

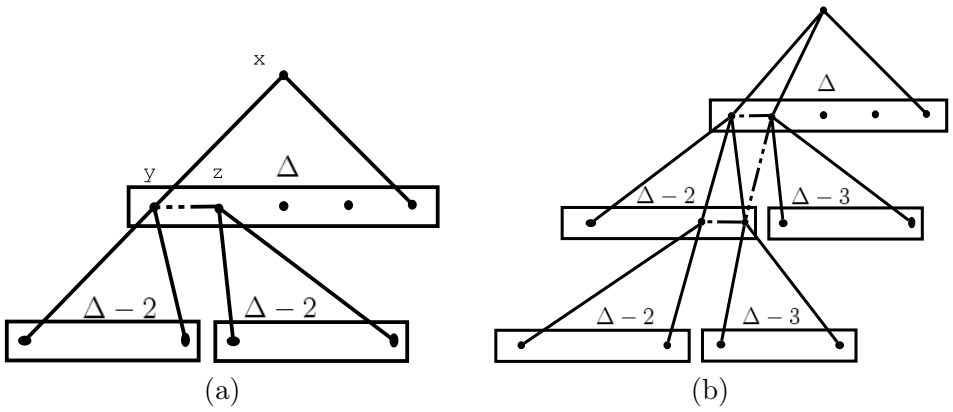

Figure 4.6: a) Lower bound. b) Greedy first-fit lower bound.

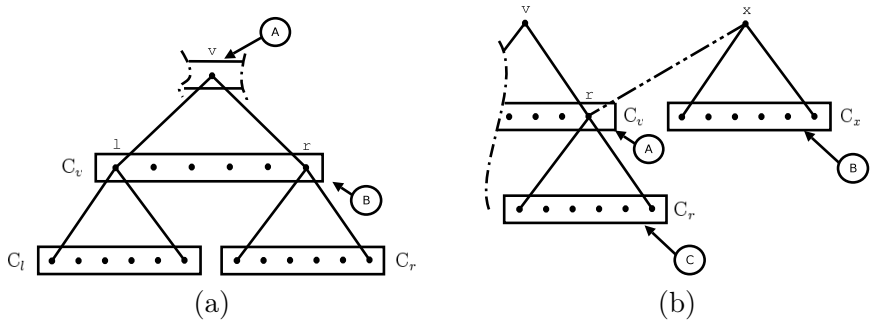

Figure 4.7: a) Color set assignment. b) Fixed right color set.

Proof: Consider the graph shown in Figure 4.6.a; x, y and z are nodes of degree Δ. As all adjacent nodes of x, y and z are at mutual distance ≤ 3, it is easy to see that it requires at least $3\Delta - 3$ colors. □

The greedy first-fit approach is a frequently used technique for labeling nodes of graphs and usually performs rather well in practice. This technique consists in considering nodes one by one in any order and assigning them the first color not used by any of their labeled neighbors satisfying the $L(1,1,1)$-labeling condition. If there is a tree-like structure, the followed order is typically the top-down left to right one. In our case, we can state the following theorem.

Theorem 4.2.3 *There exists an outerplanar graph G of degree Δ such that the greedy first-fit approach requires at least $4\Delta - 7$ colors to $L(1,1,1)$-label G.*

Proof: We refer to Figure 4.6.b. A greedy first-fit algorithm assigns label 0 to the root; labels from 1 to Δ to the root's children; labels from $\Delta+1$ to $3\Delta-5$ to the root's grandchildren, in view of the edge connecting their parents. The first $\Delta - 2$ nodes of the last level can assume colors from the set $\{3, \ldots, \Delta\}$, while the remaining nodes must use new labels from the set $\{3\Delta - 4, \ldots, 4\Delta - 8\}$. Figure 4.6.b where there are $4\Delta - 3$ nodes that are within distance 3 of a node v. □

Since the gap between the lower bound on $\chi_{1,1,1}$ and the guaranteed performance of the greedy first-fit approach is rather large, we now present an algorithm that, given an outerplanar graph G of maximum degree $\Delta \geq 6$, finds an $L(1,1,1)$-labeling of nodes of G using at most $3\Delta + 9$ colors, and hence is almost optimal as the lower bound is at least $3\Delta - 3$.

Let `A` be the color set $\{0, 1, \ldots \Delta + 2\}$, `B` the color set $\{\Delta + 3, \Delta + 4, \ldots 2\Delta + 5\}$ and `C` the color set $\{2\Delta + 6, 2\Delta + 7, \ldots, 3\Delta + 8\}$; each set has size $\Delta + 3$. The first step of the algorithm is to build an OBFT(G), rooted on a node of degree 1 or 2. Then the algorithm proceeds to assign a set of colors to the children of each node. Finally, it colors each node with a color from its color set.

Before describing how to assign sets of colors, we first introduce some definitions.

For a node v, let C_v denote the children of v in the OBFT(G) and $S(C_v)$ be the color set that is assigned to C_v. $S(v)$, where v is a single node, denotes the color set assigned to the set composed by v and its siblings. At each step we assign color sets in a way such that conflicting sets are avoided. By conflicting sets we mean that the colors in the sets may violate the $L(1,1,1)$-labeling condition.

Let v be a node assigned to a specific set of colors (refer to Figure 4.7.a). All grandchildren of v are at distance ≤ 3 from C_v, hence we must forbid set $S(C_v)$ to all grandchildren of v and, in general, we are free to choose between the two remaining sets. Since v and possibly its left and right siblings (if they are adjacent to v), are at distance ≤ 3 from the grandchildren of v, we prefer to choose the color set different from the one already assigned to v and its siblings when possible.

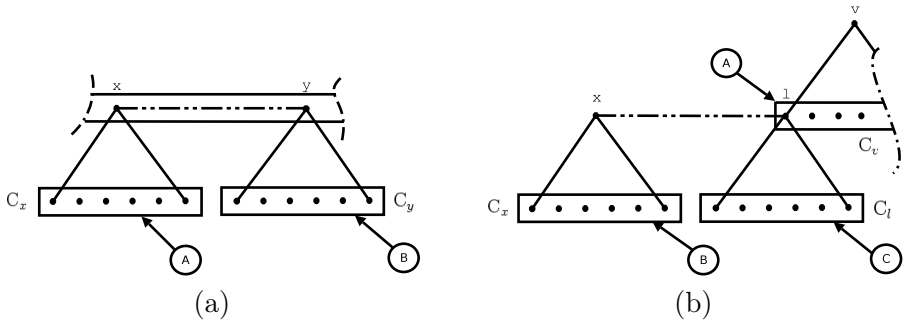

Figure 4.8: a) Alternate color sets. b) Fixed left color set.

Occasionally, we will have no choice but to assign a specific color set because it is the only color set left that can be assigned without causing conflicts. This can occur for the grandchildren of v that are children of either a leftmost or rightmost child of v. We call these color sets *fixed* (see Figure 4.7.b and Figure 4.8.b).

We now describe how to assign a color set. The color sets are assigned level by level top down from the root to the leaves and from the left to the right within each level of the tree, except in some special cases that will be explained later.

After we have assigned two separate color sets to the root and its children, we have two levels that are fully assigned and we have to assign color sets to the third level. Assume that we have already assigned color sets to level h and $h+1$, $h \geq 1$; we are now ready to assign color sets to level $h+2$. Suppose v and its children C_v have been assigned sets, (refer to Fig 4.7.a). In order to assign color sets to v's grandchildren we first have to check C_r, where r is the rightmost child of v. The only case in which we do not follow the left to right order is depicted in Figure 4.7.b: if there is non-tree edge (r,x) (i.e. the distance between any node in C_r and any node in C_x is ≤ 3) and the color sets $S(C_v) \neq S(C_x)$ then we have no choice but to assign the only color set available to C_r.

Afterwards, we have to check if node r is connected to its left sibling by a non-tree edge. If so, we have to assign sets from right to left, alternating with the only color set left available (see Figure 4.8.a), until there is a missing non-tree edge, which will occur due to Corollary 4.2.1. Next, we check the leftmost child l of node v. Again, if the color set to be assigned to C_l is fixed (Figure 4.8.b), we have to assign the only available set and then check if node l is connected via a non-tree edge to its right sibling. If so, repeat the alternating set assignment as before (until a missing non-tree edge is encountered). After the two boundary sets have been assigned, we try to assign color sets from left to right using a color set that is different from $S(v)$ if possible, alternating color sets from left to right for any non-tree edge that is present.

A more formal description is given in Algorithm 4.

Theorem 4.2.4 *There exists a linear time algorithm that $L(1,1,1)$-labels any outerplanar graph with $3\Delta+9$ colors, where $\Delta \geq 6$.*

Proof: We have already described the first two steps of the algorithm (i.e. the construction of OBFT(G) and the color set assignment), so it remains to detail how to assign to each node a color from its color set satisfying the $L(1,1,1)$-labeling condition.

Given a node group and its assigned color set, we can arbitrarily choose a different color for each node, paying attention only to nodes that are at distance ≤ 3 from a node group having the same color set. So, we first assign colors to such nodes (there are no more than four: the leftmost,

Algorithm 4 : `Assign Color Set`

Let `A`, `B` and `C` be the three sets of distinct colors, each of size $\Delta + 3$;
construct OBFT(G) tree T of an outerplanar graph G, rooted on a node of degree 1 or 2;
assign $S(\{root\})$=A and $S(C_{root})$=B;
suppose color sets have been assigned to nodes at levels h and $h+1$, $h \geq 1$;
visit nodes of T top down, left to right starting from the root;
for each node v on level $h \geq 1$ **do**
  let r be the rightmost child of v and l be the leftmost;
  if $S(C_r)$ is *fixed* (see Figure 4.7) **then**
    assign the only available color set to C_r;
    proceed right to left (see Figure 4.8.a), assign color set to C_x alternating between the two available until there is a missing non-tree edge between x and y, or until a color set has beed assigned to C_l;
  end if
  if $S(C_l)$ is *fixed* (see Figure 4.8) **then**
    assign the only available color set to C_l ;
    proceed left to right (see Figure 4.8.a), assign color set to C_y alternating between the two available until there is a missing non-tree edge between x and y, or until a color set has beed assigned to C_r ;
  end if
  let z be the leftmost node in C_v such that $S(C_z)$ is not assigned;
  while there is such a node z **do**
    assign to C_z the available color set not assigned to the parent of z;
    proceed from left to right alternating color sets as in Figure 4.8.
  end while
end for

its right sibling, the rightmost and its left sibling) avoiding conflicts, and then we proceed with all other nodes.

It is straightforward to see that this algorithm correctly labels the graph in linear time. It remains to show that $\Delta + 3$ colors in each group are always enough.

Let us fix any node x on an OBFT(G) and its set of children C_x. Without loss of generality, let our algorithm assign color set `A` to C_x . It is easy to see that the worst case for the cardinality of `A` is when C_x is at distance ≤ 3 from as many nodes as possible, all colored with a color in `A`. This happens when there are as many non-tree edges as possible, as they somehow shorten the distances computed on the tree. For this reason, let x be the rightmost sibling as this configuration allows the presence of non-tree edge (s, m) (refer to Figs. 4.9.a and 4.9.b for the notation) .

Two cases are possible, according to the existence of non-tree edge (x, y).

According to the algorithm, if such an edge exists (see Figure 4.9.a) both C_x and the group of nodes to which t belongs to receive the same color set `A`. In order to maximize the number of nodes at distance ≤ 3 from C_x, let us consider the case in which the algorithm assigns color set A to the group to which y belongs to and to the group of nodes children of the left sibling of x. It is easy to see that, according to our algorithm, no other nodes at distance ≤ 3 can receive a color from the color set `A`. Hence, exactly $|C_x|$ nodes must be labeled using colors from `A`, avoiding the color assigned to nodes t, t', y y' and l. Since $|C_x| \leq \Delta - 2$, $\Delta + 3$ colors in `A` are sufficient.

If edge (x, y) does not exist the algorithm assigns color sets as shown in Figure 4.9.b. (note that the group of nodes to which m belongs to has a fixed left color set due to the non-tree edge (t, y)). Hence, nodes in C_x must be labeled using colors from set `A` avoiding the colors assigned to nodes

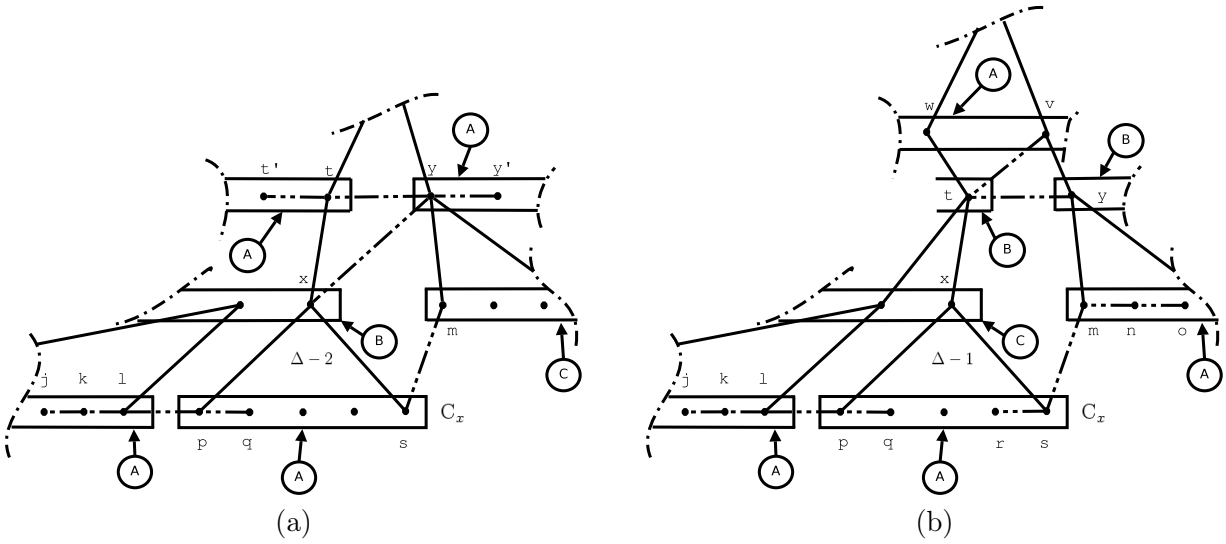

Figure 4.9: Proof of Theorem 4.2.4.

m, l, w and v all at distance ≤ 3 from nodes in C_x. Since $|C_x| \leq \Delta - 1$, $\Delta + 3$ are sufficient.

Furthermore, observe that the color assigned to j cannot be used in p, the color assigned to k cannot be used neither in p nor in q, and similarly the color assigned to o cannot be used in s and the color assigned to n cannot be used neither in r nor in s. It follows that, after removing the 4 colors forbidden by m, l, w and v, the $\Delta - 1$ remaining colors must be at least 4. In the special case in which the color assigned to k is the same as the color assigned to n, one color more is necessary. Hence we need the precondition $\Delta \geq 6$. □

We conclude this subsection observing that if $\Delta \leq 6$, the algorithm requires anyway 27 colors to perform the labeling.

4.2.5 L(h,1,1)-Labeling

In this subsection we show how to generalize the results of the $L(1,1,1)$-labeling to the $L(h,1,1)$-labeling for $h \geq 2$.

First, observe that Theorem 4.2.2 provides a lower bound of $3\Delta - 3$ for $\chi_{h,1,1}$, for any $h \geq 1$. Also Theorem 4.2.3 on the greedy first-fit approach applies to the general case $h \geq 1$.

In the following, we detail how to get an $L(h,1,1)$-labeling by exploiting Algorithm `Assign Color Set` and then by labeling nodes with colors from the assigned set. Color sets are separated by a gap in order to address the requirement of spacing adjacent nodes by at least h colors apart. More precisely, set `A` contains colors $\{0,1,\ldots\Delta+2\}$, set `B` colors $\{\Delta+h+2, \Delta+h+3, \ldots 2\Delta+h+4\}$ and set `C` colors $\{2\Delta+2h+4, 2\Delta+2h+5, \ldots, 3\Delta+2h+6\}$. The $h-1$ colors in the gaps between color sets guarantee that the distance 1 constraint between adjacent groups of nodes is respected.

As a building block for $L(h,1,1)$-labeling outerplanar graphs, we need to be able to perform a labeling of paths as stated in the following lemma.

Lemma 4.2.2 *Given any integer $h \geq 2$, it is possible to label with $l \geq 2h+1$ consecutive colors any path having at most l nodes, respecting the following constraints:*

- *each color must be assigned to at most one node;*
- *adjacent nodes must receive colors that are at least h apart.*

Figure 4.10: Path labeling: a) consecutive colors, b) colors with holes.

Proof: Without loss of generality, suppose we have colors from 0 to $l-1$. Assign to the first node of the path any color x, to nodes in position $2i, i = 1, \ldots, \lfloor \frac{l}{2} \rfloor$ color $(x + \lfloor \frac{l-1}{2} \rfloor + i) \bmod l$, to nodes in position $2j-1, j = 1, \ldots, \lceil \frac{l}{2} \rceil$ color $(x + j - 1) \bmod l$ (see Figure 4.10.a). It is easy to see that this labeling respects the constraints if and only if $l \geq 2h+1$. Moreover, this labeling has minimum span.

□

Remark 4.2.1 *Modifying the algorithm described in the proof of Lemma 4.2.2 in a way such that it assigns the i-th color in an ordered list of $l \geq 2h+1$ non necessarily consecutive colors, it will still find a valid assignment for a path of length l (see Figure 4.10.b).*

Note that once we have assigned a color set to each group of siblings, each such group induces a subgraph of a path. Moreover, we already know that in the worst case scenario, we will have $\Delta - 1$ nodes to label with exactly $\Delta - 1$ available colors. Nevertheless as it may happen that the end points on both sides of the resulting path are not freely choosable, we need some refinement in order to apply Lemma 4.2.2.

Consider the worst case scenario depicted in Figure 4.11, where the $\Delta - 1$ children of node x have to be labeled with the remaining $\Delta - 1$ colors. In such a configuration, in view of Corollary 4.2.1, at least one non-tree edge connecting two siblings must be missing. Consider the path from $l1$ to the leftmost node missing its right non-tree edge and the path from $r1$ to the rightmost node missing its left non-tree edge; without loss of generality, let the path starting from $r1$ be the shortest one. We label first $r1$ using the first available color, keeping into account the constraints induced by already colored nodes. Then we label $r2$ with a color at least h apart from the color assigned to $r1$. Now we complete the labeling of the path by using Lemma 4.2.2.

It is not restrictive to assume that all the remaining uncolored siblings constitute a unique path, as otherwise the constraints are weaker; so we repeat the same procedure to label the path starting from $l1$. Observe that the produced labeling is feasible and we are always able to perform it, provided that enough colors are available.

Theorem 4.2.5 *For any $h \geq 2$, there exists a linear time algorithm that $L(h,1,1)$-labels any outerplanar graph with $3\Delta + 2h + 7$ colors if $\Delta \geq 4h + 7$.*

Proof: As Algorithm `Assign Color Set` does not depend on h, we have already proved in Subsection 4.2.4 that it can be run successfully, guaranteeing that at least $\Delta - 1$ colors are always available to label at most $\Delta - 1$ siblings. The claim is proved if we show that the available colors are always enough to complete the labeling of each group of siblings.

Refer to Figure 4.11. First observe that the labeling of the paths starting from $l1$ and $r1$ are subject to equivalent constraints, and that when we label the first path we have much more colors to chose from, so it is enough to prove that the remaining colors are sufficient to label the second path.

To label the first node, $l1$, at most $2h$ (distance 1 from a) plus 2 (distance ≤ 3 from b and c) colors cannot be used. The shortest path consists of at most $\lfloor \frac{\Delta-1}{2} \rfloor$ nodes, so there is at least one color available as $\Delta - 1 - \lfloor \frac{\Delta-1}{2} \rfloor - 2h - 2 \geq 1$ when $\Delta \geq 4h + 7$.

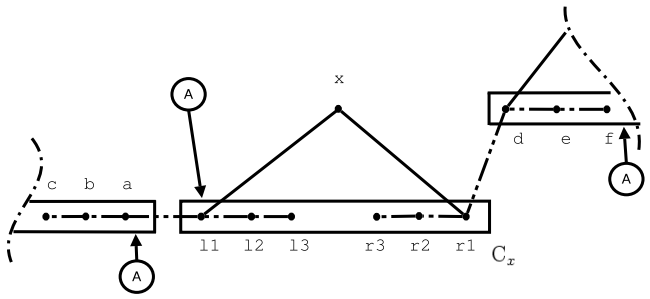

Figure 4.11: $L(h, 1, 1)$-labeling worst case.

In order to label $l2$, we have at least $\Delta - 1 - \lfloor \frac{\Delta-1}{2} \rfloor - 1$ colors, from which we have to eliminate at most $2h$ (distance 1 from $l1$) $+1$ (distance 3 from b) colors, so at least $\lceil \frac{\Delta-1}{2} \rceil - 1 - 2h - 1 \geq 1$ colors. Finally, $l2$ is also the first node of the path labeled using Lemma 4.2.2; hence we must prove that there are at least $2h + 1$ available colors. This is always true as, among the $\Delta - 1$ colors, we used at most $\lfloor \frac{\Delta-1}{2} \rfloor$ to label the shortest path and 1 color to label $l1$, so the remaining colors are at least $2h + 2$. Observing that the length of the longest path never exceeds the number of available colors, the claim follows.

□

We conclude this subsection by observing that even in the general case of the $L(h, 1, 1)$-coloring, there is a threshold for values of the maximum degree; in this case, if $\Delta \leq 4h + 7$, the algorithm requires anyway at most $14h + 28$ colors.

4.2.6 Conclusion

In this section we provided very close upper and lower bounds on the number of colors, $\chi_{h,1,1}$, that are needed to $L(h, 1, 1)$-label an outerplanar graph for $h \geq 1$. We showed that the greedy first-fit technique does not work well in this case. In the literature there is an algorithm that optimally $L(1, 1, 1)$-labels outerplanar graphs running in time $O(n^3)$ [169], but the multiplicative factor is too large to be of practical use. Our algorithm produces an approximate solution that only differs from the optimal solution by a constant additive factor, and it is linear.

Some open problems arise from this work. First, there is a gap between the upper bound provided by the algorithm and the lower bound shown. It would be nice to close the gaps between the bounds.

Furthermore, the upper bounds we found are rather large for small values of Δ, and can probably be improved: our aim was to find an algorithm with a good asymptotic behavior.

Finally, for $L(h, 1^d) = L(h, 1, \cdots, 1)$, we have only studied the case when $d = 2$. It would be interesting also to study the $L(h, 1^d)$-labeling problem of outerplanar graphs for $d \geq 3$. The same technique of using color group assignments can be applied, but the number of cases to be considered increases quite a bit. The problem here is to find good estimates for $f(h, d)$ and $g(h, d)$ in the inequality $\chi_{h,1^d} \leq f(h, d)\Delta^{\lceil \frac{d}{2} \rceil} + g(h, d)$.

4.3 Acknowledged Broadcasting in Ad Hoc Radio Networks

4.3.1 The Model and the Problem

A radio network consists of stations that can transmit and receive messages. Stations have synchronized clocks showing the same round number. The network is modeled as a **directed graph** with a distinguished node called the *source*. If there is a directed edge from u to v, node v is called an *out-neighbor* of u and u is called an *in-neighbor* of v. In a given round each station can act either as a transmitter or as a receiver. Two alternative assumptions are made about radio networks: either nodes can distinguish collision from silence or they cannot do it. In the first case we say that collision detection is available, in the second case - that it is not.

We assume that each node knows only its own label but is unaware of the topology of the network, or of any bound on its size or diameter. Labels of a n-node network are distinct integers from the set $1, \ldots, N$, where N is $O(n)$. To simplify notation, we will assume that $N = n$ but all our results hold without this additional assumption.

We consider only deterministic communication algorithms and do not assume any central monitor of the broadcasting process. Thus the decision made by a node on whether to transmit or to receive in a given round, and what message to transmit, if any (some control messages can be transmitted on their own or be appended to the source message) is based exclusively on the label of the node, the round number, and on the messages it heard so far.

Acknowledged broadcasting is a communication task consisting in transmitting a message from the source to all other nodes of the network and making this fact common knowledge among all nodes. Acknowledged broadcasting was defined in [43] as the task of completing broadcasting and informing the source about it. More precisely, there must exist a round t at which all nodes know the source message and the source is aware of this. Notice that after round t the source may broadcast another message "*done at round* $2t$", using the same algorithm. This message is guaranteed to reach all nodes by round $2t$, and precisely at this round the source message becomes *common knowledge* [101] in the network, i.e., all nodes know it, all nodes know that all nodes know it, all nodes know that all nodes know that all nodes know it, etc. (arbitrarily many times). Since this common knowledge can be achieved in twice the time needed to inform the source that broadcasting is completed, both tasks have the same time complexity. We will use the term *acknowledged broadcasting* and the abbreviation AB, for the stronger task. However, the results we present apply to both versions of AB.

The execution time of a broadcasting algorithm in a given radio network is the smallest round number after which all nodes of the network have heard the source message and no other messages are sent. Likewise, the execution time of an AB algorithm is the smallest round number after which all nodes of the network have common knowledge of the source message. Having this information they do not transmit any further messages. We recall that in the spontaneous wake up model, it was proved in [43] that AB is infeasible without collision detection, even for symmetric networks. On the other hand, assuming collision detection, the authors provided an AB algorithm working in time $O(nD)$, for arbitrary strongly connected n-node networks, where D is the eccentricity of the source (the maximum length of all shortest paths in the graph from the source to all other nodes). Uchida et al. [161] showed an AB algorithm without collision detection working in time $O(n^{4/3} \log^{10/3} n)$ for all strongly connected networks of size at least 2.

4.3.2 Original Contribution

We improve the above results from [43] and [161] by presenting two generic AB algorithms for the spontaneous wake up model using a broadcasting algorithm without acknowledgement, as a procedure. For a large class of broadcasting algorithms the resulting AB algorithm has the same

time complexity. Using the currently best known broadcasting algorithms, we obtain an AB algorithm with collision detection working in time $O(\min\{n \log^2 D, n \log n \log\log n\})$, for arbitrary strongly connected networks, and an AB algorithm without collision detection working in time $O(n \log n \log\log n)$ for all strongly connected networks of size $n \geq 2$. Moreover, we show that in the conditional wake up model, AB is infeasible in a strong sense: for any AB algorithm there exists an infinite family of networks for which this algorithm is incorrect.

4.3.3 AB in the Spontaneous Wake Up Model

Availability of Collision Detection

Here we present an AB algorithm `AckBroadcast1` working correctly for all strongly connected n-node networks with source eccentricity D, where n and D are arbitrary and unknown. We assume the spontaneous wake up model with collision detection. The algorithm works in four stages. The first two of them are the same as in Algorithm Bound-Broadcast-Check from [43]: in the first stage all nodes learn an upper bound on n within a factor of 2, and in the second stage each node learns its distance from the source. The rest of the algorithm contains a crucial difference which permits us to improve the time $O(nD)$ of Algorithm Bound-Broadcast-Check to the currently best known time of broadcasting without acknowledgement, which is the minimum of $O(n \log^2 D)$ [56] and $O(n \log n \log\log n)$ [57]. Instead of interleaving phases of broadcasting and phases of acknowledgement, as it was done in Algorithm Bound-Broadcast-Check and which was a time-consuming procedure, we first try to learn an approximate value of D and only then broadcast. More precisely, in Stage 3 all nodes learn an upper bound on D within a factor of 2, and in Stage 4 broadcasting is performed, using approximate knowledge of n and D to time-out the process. A more detailed description of the algorithm follows. We use the following terminology, introduced in [43]. A node v acting as a receiver in a given round *hears signal* μ, if at least one of its in-neighbors acts as a transmitter, i.e., if v hears a message or if there is a collision at v in this round. Otherwise (if no in-neighbor of v acts as a transmitter), v hears nothing. A *contact message* is a fixed one-bit signal.

Algorithm `AckBroadcast1`

Stage 1. This stage proceeds in phases numbered by consecutive positive integers. The kth phase lasts $2^k + 1$ rounds, numbered $1, \ldots, 2^k + 1$, and each phase immediately follows the preceding one. A node is *active* after the jth phase, $j > 1$, if it acted as a transmitter in some round of this phase. All nodes are active in the first phase. It was proved in [43] that after each phase either all nodes are active or all are not active. In the first round of phase k all active nodes with labels larger than 2^k send a contact message and all others act as receivers. In round $i > 1$ of the kth phase, each active node that heard signal μ in round $i - 1$ of this phase and has not heard it in any previous round of this phase, sends a contact message. All other nodes act as receivers. The first phase after which all nodes are not active is the last phase of the first stage. Each node can identify this phase as the first phase after which it is not active. Call this phase j_0. At this point all nodes know that $2^{j_0-1} < n \leq 2^{j_0}$. Let $x = 2^{j_0}$. At the end of Stage 1 every node knows j_0 and hence knows x.

Stage 2. This stage is initiated by the source immediately following phase j_0 of Stage 1. This stage lasts x rounds. In round 1 of Stage 2 the source acts as a transmitter sending the contact message, and all other nodes act as receivers. In round $i > 1$ those nodes that heard signal μ in round $i - 1$ act as transmitters, sending a contact message, and all others act as receivers. Each node learns its distance from the source as the round number of Stage 2 in which it heard the signal μ for the first time. Each node knows when Stage 2 ends.

Stage 3. This stage starts immediately after the end of Stage 2 and proceeds in phases numbered

by consecutive positive integers. Each phase lasts x rounds. In round 1 of phase k all nodes whose distance from the source is larger than 2^k transmit a contact message and all others act as receivers. In round $i > 1$ of the kth phase, each node that heard signal μ in round $i-1$ of this phase and has not heard it in any previous round of this phase, sends a contact message. All other nodes act as receivers. The first phase r in which the source does not hear the signal μ is the last phase of this stage. Each node can identify this phase as the first phase in which it has not heard the signal μ. At this point all nodes (and in particular the source) know that $2^{r-1} \leq D < 2^r$.

Stage 4. Immediately following the end of Stage 3 starts a broadcasting algorithm for ad hoc networks with known parameters x (the upper bound on the number of nodes), $y = 2^r$ (the upper bound on the eccentricity of the source), and known upper bound $T(x, y)$ for completion of broadcasting for all networks with these parameters. Every node waits until the $(T(x, y))$th round of Stage 4. At this point common knowledge of the source message is achieved. ■

Remark. If the broadcasting algorithm used in Stage 4 is such that $T(x, y)$ depends only on x and not on y, (e.g., the $O(n \log n \log\log n)$ algorithm from [57]), Stage 3 can be deleted, with Stage 4 following Stage 2.

Theorem 4.3.1 *There exists an AB algorithm working correctly for all strongly connected networks with collision detection, and using time in $O(\min\{n \log^2 D, n \log n \log\log n\})$, for networks with n nodes and source eccentricity D.*

Proof: Stage 1 takes time $O(x) = O(n)$. Stage 2 takes time $x \in O(n)$. Stage 3 takes time $O(xr) = O(n \log D)$. Stage 4 lasts as long as the broadcasting algorithm used for ad hoc networks with (approximate) parameters learned in the previous stages. Taking the best of the algorithms $O(n \log^2 D)$ [56] or $O(n \log n \log\log n)$ [57] for the given pair of parameters implies the claimed result. (Knowing both parameters within a factor of 2 does not change the final complexity). □

Non-Availability of Collision Detection

Here we present an AB algorithm working correctly for all strongly connected n-node networks without collision detection, where $n \geq 2$ is arbitrary and unknown. We assume the spontaneous wake up model without collision detection. Our algorithm works in the currently best known time of broadcasting without acknowledgement for networks of unknown size n, which is $O(n \log n \log\log n)$ [57]. We present the algorithm in a generic form, using as a procedure an arbitrary algorithm $\mathcal{B}(r)$ for broadcasting (without acknowledgement) in networks without collision detection, of size at most r, with a known upper bound $B(r)$ on completion time for such networks.

Algorithm `AckBroadcast2`

The algorithm proceeds in phases: each phase consists of four stages. During phase i, we call *small* a node with a label at most 2^i and *big* a node with a label greater than 2^i. Fix a phase $i \geq 1$.

Stage 1. This stage consists of 2^i rounds. Each small node transmits its label once in a round-robin fashion: the node with label x transmits in the xth round of the stage. Big nodes transmit during all rounds of this stage. At the end of the stage, each small node without big in-neighbors, knows the complete set of its in-neighbors (notice that this set cannot be empty for a small node with this property because the network is strongly connected and has at least two nodes). Small nodes with at least one big in-neighbor, either know the label of their in-neighbor, if it is unique, or hear silence for the whole duration of the stage, otherwise. At the end of this stage, each small node knows whether it has a big in-neighbor.

Stage 2. Apply algorithm $\mathcal{B}(2^i)$. Small nodes participate in it, while big nodes are silent. A small node which receives the source message is called *informed*. The second stage lasts $B\left(2^i\right)$ rounds.

Stage 3. Small nodes which are either uninformed or aware of a big in-neighbor, transmit a *failure* message, in a round-robin fashion. This stage lasts 2^i rounds.

Stage 4. This stage lasts $B\left(2^i+1\right)$ rounds. Only small informed nodes participate (all other nodes are silent). The stage consists in disseminating the failure message, from all nodes aware of the failure, to all informed nodes. We show in Lemma 4.3.1 that this can be done in $B\left(2^i+1\right)$ rounds.

If the source receives the failure message during the fourth stage of phase i, then phase $i+1$ is initiated, otherwise broadcasting is acknowledged: all nodes get common knowledge of the source message by time-out, if they obtain no failure message by the $\left(B\left(2^i+1\right)\right)$th round of Stage 4. (Any node that did not get the failure message by this round knows that no node got the failure message.) In this case the algorithm ends. ■

Lemma 4.3.1 *Any algorithm performing broadcast for all n-node networks within time $B(n)$ can be modified to disseminate the same message from multiple sources in any n-node network, within time $B(n+1)$.*

Proof: The fact that disseminating the same message from more than one source is not more difficult than broadcasting from a single source has been stated in [49, 50]. For the sake of completeness, we provide a proof of this assertion (cf. [88]).

Consider any algorithm A, completing broadcasting on all n-node networks within time $B(n)$. Consider a n-node network N, not necessarily strongly connected. We are given a set $S = \{s_1, \ldots, s_k\}$ of nodes in N, chosen by an adversary respecting the condition that each node in N is reachable from some node in S (as otherwise disseminating a message from S would be impossible). Nodes in S share a message m to be disseminated to the whole network. Now consider the network N', obtained by adding a new node s' to N and all oriented edges (s', s_i), for $1 \leq i \leq k$. Every node in N is reachable from s', so algorithm A must complete broadcast from s' in N' within time $B(n+1)$. As s' has no in-neighbor, its communication pattern during the execution of algorithm A is independent of the network N, and thus it is completely predictable once A is known. It follows that an algorithm A', disseminating message m from S in time at most $B(n+1)$, can always be obtained from algorithm A by simulating the actions of algorithm A for nodes in S in the presence of the fixed communication pattern of node s', and executing algorithm A on all other nodes in N. □

The following two lemmas establish the correctness of Algorithm `AckBroadcast2`.

Lemma 4.3.2 *If broadcast in Stage 2 of phase i of Algorithm* `AckBroadcast2` *informs all nodes in the network, then this fact becomes common knowledge by the end of phase i.*

Proof: For broadcasting to be successful in phase i, no node in the network can have a label larger than 2^i, thus no failure can be signalled. This fact becomes common knowledge at the end of phase i. □

Lemma 4.3.3 *If broadcast in Stage 2 of phase i of Algorithm* `AckBroadcast2` *fails to inform all nodes in the network, then all nodes in the network become aware of it by the end of phase i.*

Proof: Any node with a label larger than 2^i knows that phase i will be unsuccessful (by looking at its label). All nodes with an in-neighbor with label larger than 2^i become aware of failure of phase i during the first stage of this phase. Small nodes which remain uninformed at the end of stage 2 of phase i, learn that the phase is unsuccessful by time-out (because they don't get the source message). In order to prove that all other nodes become aware of the failure by the end of phase i, we will show that after the third stage of phase i, the failure message is available to some

node inside the strongly connected component C, containing the source s, in the subgraph induced by informed small nodes. This is enough to prove that dissemination of the failure message can be performed in the subgraph induced by informed nodes. The latter, together with Lemma 4.3.1, implies that all nodes become aware of the failure within time $B(2^i + 1)$.

Let x be a node aware of the failure before the third stage of phase i, and let $p = x \rightsquigarrow p_1 \rightsquigarrow \ldots \rightsquigarrow p_k \rightsquigarrow s$ be a shortest path from x to the source s. We can assume that all internal nodes in p are small in phase i, as otherwise we can repeat the reasoning by taking instead of x, the big internal node in p which is the closest to s. We can also assume that all nodes in p are informed, as if a node p_j in p is uninformed, then it is aware of the failure, and again we can use such a node p_j instead of x. Node p_1 receives the failure message from x during the execution of the third stage of phase i. As p_1 is informed, there must be a path from s to p_1 containing only small nodes in phase i. On the other hand, there is a path from p_1 to s containing only small nodes, and thus p_1 is in C. If the path p contains only s and x, then the source s has x as an in-neighbor and thus it becomes aware of the failure by the time of completion of the third stage of phase i. □

Theorem 4.3.2 *There exists an AB algorithm working correctly for all strongly connected networks of size at least 2 without collision detection, and using time $O(n \log n \log\log n)$ for networks with n nodes.*

Proof: Stages 1,2, and 3 of phase i take time 2^i and Stage 4 takes time $B\left(2^i + 1\right)$. Since $B\left(2^i + 1\right) \geq 2^i$, phase i takes time $O(B\left(2^i + 1\right))$. Taking the broadcasting algorithm from [57] with $B(r)$ in $O(r \log r \log\log r)$ gives the claimed result. □

Remark 4.3.1 *It should be noted that (as opposed to Algorithm* `AckBroadcast1`*) Algorithm* `AckBroadcast2` *cannot guarantee time $O(n \log^2 D)$ because nodes do not learn any approximation of the source eccentricity D. In fact, learning such an approximation without collision detection seems to be much more time-consuming than when collision detection is available. Thus, for shallow networks (with D polylogarithmic in n), Algorithm* `AckBroadcast1` *is (slightly) faster than Algorithm* `AckBroadcast2`.

4.3.4 Impossibility of AB in the Conditional Wake Up Model

In this section we prove that performing AB for all strongly connected networks is impossible in the conditional wake up model. As opposed to the impossibility result from [43] for the spontaneous wake up model, our negative result is not linked with any particular network (the singleton network in the case of [43]) but holds in a strong form: we show that any AB algorithm is incorrect for infinitely many networks.

Theorem 4.3.3 *For any AB algorithm there exists an infinite family of strongly connected networks for which this algorithm is incorrect.*

Proof: Consider any AB algorithm $\mathcal{A}$ and let m be an arbitrary positive integer. Let C be the m-node clockwise oriented cycle. Suppose that $\mathcal{A}$ takes time x to be completed on C. Let v be a node in C different from the source. Attach to v an oriented cycle C' of length $x + 1$ that has only the node v in common with C. Call the resulting network Q. Let w be the node preceding v in the cycle C'. Node w can be woken up in round $x + 1$ at the earliest. Hence by round x the behavior of all nodes belonging to C must be identical in networks C and Q. Since algorithm $\mathcal{A}$ completes AB in network C by round x, the nodes in C must also complete AB in network Q by round x. In particular, at round x the source in Q considers broadcasting to be completed in network Q, which is false because node w has not yet obtained the source message at this point. It follows that $\mathcal{A}$ is

incorrect either in C or in Q. Since both these networks have size at least M, this concludes the proof. □

4.3.5 Conclusion

In this section we proved that acknowledged broadcasting can be completed with the same time complexity of the best known broadcasting algorithms without acknowledgment, both with and without collision detection (in the latter case the network must contain at least two nodes).

An open question remains in the case when collision detection is not available. As stated in Remark 4.3.1, in this setting we are unable to take advantage of the $O(n \log^2 D)$ time algorithm from [56] for networks of small diameter, as we are unable to estimate the diameter of the network fast enough.

Chapter 5

Fault Tolerance

5.1 Introduction

Addressing communication problems over networks it is fundamental to consider reliability. The vulnerability to faults of a communication network grows with its size; the higher is the number of components, the higher is the probability that some of them are defective.

Modern networks are composed of a huge number of nodes. Large scale sensor networks are expected to contain several thousands of devices; millions of cellular phones - which are quickly evolving from single purpose, voice communication devices, to general purpose machines, with fairly good computational and memory capabilities - are constantly connected in a network. Modern supercomputers, built from leading companies like IBM, are massively parallel machines, with tens of thousands processing units operating over an interconnection network. Even cheap desktop computers are turning into parallel machines, with the wide diffusion of multi-core architectures for the processing units.

It is thus evident that fault tolerance capabilities are becoming more and more important. Fault tolerant algorithms and architectures indeed are needed in order to keep this impressive plethora of networks functional and effective.

The models developed to study the problem of faults are in constant evolution. Faults may happen on nodes or on links, and they can be transient or permanent. The types of faults that are more widely considered in the literature are *crash* faults and *Byzantine* faults. Crash faults model hardware failures, either at node level or at link level; faulty nodes stop transmissions and faulty links do not deliver the messages sent over them, thus the functionality of the fault-free part of the network is not altered. In the Byzantine fault model, faulty nodes and links can instead actively operate to tamper the functionality of the fault-free ones. This kind of faults hardly arises from hardware or software malfunctioning, but represents a useful method to investigate the vulnerability of the network to hostile agents.

Different models have been developed for what concerns the distribution of faults over the network as well: faults have to be constrained somehow, as if all the components of the network are faulty, nothing can be done in order to complete any meaningful task. The first models for fault distribution where mostly combinatorial: algorithms and architectures capable of tolerating up to a given number k of faulty components of the networks where sought. Correctness of such solutions has to be guaranteed in the worst case. A probabilistic model for fault distribution gained popularity as it represents more faithfully the behavior of real networks, where worst case distributions of faulty components have a very small probability to arise. A slightly outdated but still interesting survey on the subject of fault tolerance (focusing on fault tolerant broadcast and gossip) can be found in [144].

The first probabilistic models developed to study fault distributions assigned a given probability to network components to be (or become) defective, thus faults where distributed randomly and independently. Recent efforts have been made to develop probabilistic fault distribution models where the failure probability of close nodes are correlated (see [123], for an analytic study of such a model and references to other empirical works which justify this choice in modeling).

A different approach to handling faults is the one of diagnose and repair. Self-stabilizing systems [60, 61] can be considered an optimistic approach to fault tolerance [93], which can be very effective when faults are transient. The ability to correctly operate in spite of the presence of faults indeed, in algorithms like in architectures, is mainly based on redundancy and thus can considerably increase costs and reduce the efficiency. When temporary failures of the system are acceptable it may be more cost effective to (automatically) perform reconfigurations (self healing) after detection of a fault than relying on fault tolerance.

In this chapter we consider both approaches to fault tolerance. The contribution we present in

Section 5.2 considers the problem of self deployment of mobile sensors. Tolerance to faults in this case is considered with respect to sensing coverage of the area of interest and radio connectivity of sensing devices. We propose two variations of our self deployment protocol, one exhibiting self healing capabilities, and the other providing guaranteed tolerance to a given number of faults, where this number depends on the sensors available and the extention of the area of interest. Redundant sensors are either used to perform reconfigurations after detection of a fault or uniformely spread over the area of interest in order to guarantee sensing coverage and radio connectivity in presence of faults.

In Section 5.3 we focus on self healing properties of a network where nodes are connected in a 2-dimensional grid. We assume that each internal node in the grid can communicate with its left, right, top, and bottom neighbor directly, but some nodes can be defective. When defective nodes are detected, the network must be reconfigured, in a way such that defective nodes are simulated by correctly working ones and the functionality of the network is restored. As in this case no redundancy is available, we focus on reconfigurations of fault free nodes having the goal to minimize the performance loss.

5.2 Autonomous Deployment of Mobile Sensors

5.2.1 The Model and the Problem

We are given an area of interest (in short AoI) with arbitrary shape (the only constraing beeing that the AoI has to be connected). n sensors are arbitrarily deployed in this AoI; sensors are homogeneous devices, described by a unique label, a transmitting range, a sensing range, and a moving speed.

We do not assume any central monitor; each sensor is capable of sensing its position and knows its label. Moreover, each sensor knows the boundaries of the AoI and the total number of deployed sensors. We also assume that all sensors are equipped with synchronized clocks. We do not assume a subdivision of time in rounds, but we rely on such clocks in order to produce time-stamps. No other knowledge is initially provided to nodes, which have to coordinate themselves in order to achieve complete sensing coverage of the AoI and network connectivity.

Our aim is to provide a fully distributed protocol to coordinate the movements of sensors, trying to minimize and balance the travelling distance of each device (in order to minimize power consumption and maximize the lifetime) and trying to achieve a uniform coverage of the AoI.

We develop such protocol at the application level of the OSI stack, i.e., we assume each message sent from a sensor at a given time is correctly heard by all sensors within its transmitting range.

5.2.2 Related Work

There is an impressively growing interest in self-managing systems, starting from several industrial initiatives from IBM [107], Hewlett Packard [105] and Microsoft [137]. Various approaches have been proposed to self-deploy mobile sensors although few of them can be actually considered autonomic.

The principles of autonomic computing [9], require the proposed solutions to exhibit some basic self-* capabilities, i.e., self-configuration, self-adaptation and self-healing, while most of the existing approaches require fine tuning of arbitrary parameters and thus external human intervention.

The virtual force approach (VFA) [40, 102, 170] models the interactions among sensors as a combination of attractive and repulsive forces. This approach requires the definition of thresholds to determine the magnitude of the force one sensor exerts on another. As shown in [40], the VFA presents oscillatory sensor behavior. This problem is addressed by defining further arbitrary thresholds as stopping conditions. The tuning of such thresholds is laborious and relies on an off-line configuration. In addition, it influences the resulting deployment, the overall energy consumption and the convergence rate. Moreover, this approach does not guarantee the coverage in presence of narrows in the AoI.

A variation of the VFA is presented in [147] where the introduction of two virtual forces guarantees better uniformity by providing at least k neighbors to each sensor. Other approaches are inspired by physics as well, such as [141] and [113]. In [141] the sensors are modelled as particles of a compressible fluid and regulates their movement by mimicking a diffusive behavior. In [113] two approaches that make use of gas theory to model sensor movements in presence of obstacles are proposed. However the last three approaches still suffer from oscillatory sensor behavior.

The Voronoi approach ($\text{VOR}_{\texttt{MM}}$) is detailed in [164]. According to this proposal, each sensor iteratively calculates its own Voronoi polygon, determines the existence of coverage holes and moves to a better position if necessary. In this approach the relation between the transmitting and the sensing range influences the obtained performances by either moving sensors toward already covered positions or reducing the resulting covered area. Furthermore, this approach is not designed to improve the uniformity of an already complete coverage. According to [134] each sensor makes a rough evaluation of the local density and calculates the movements needed to reach a final position that is as close as possible to the points of a hexagonal tiling. This approach suffers from similar

limitations to the VFA and does not guarantee oscillation avoidance if proper threshold parameters are not set.

In [163] the authors analyze the problem of sensor deployment in a hybrid scenario, with both mobile and fixed sensors in the same environment. They introduce the general concept of logical movements. Instead of moving iteratively, sensors calculate their target locations based on a distributed iterative algorithm, move logically, and exchange new logical locations with their new logical neighbors. Actual movement only occurs when sensors determine their final locations, thus sparing energy by avoiding zig-zag motions at the expense of some more messaging activity.

5.2.3 Original Contribution

The main contribution of this section is a fully distributed algorithm for mobile sensor deployment called `Push&Pull`. Our approach differs from the ones of the previously cited contributions, and has the merit of being simple and effective. Its design follows the grassroots approach [9] to autonomic computing. Self-organization emerges without the need of external coordination or human intervention as the sensors autonomously adapt their position on the basis of their local view of the surrounding scenario. This way our algorithm shows the basic self-* properties of autonomic computing of self-configuration, self-adaptation and self-healing.

Algorithm `Push&Pull` produces a hexagonal tiling by spreading sensors out of high density regions and attracting them towards coverage holes. Decisions regarding the behavior of each sensor are based on locally available information and do not require any prior knowledge of the operative scenario nor any manual tuning of key parameters.

We formally prove that our algorithm terminates and provides a complete coverage regardless of the particular shape of the area of interest; moreover, we propose a variant that exploits redundant sensors to produce a k-coverage, where k depends on the number of the available sensors and on the shape and extension of the AoI.

We ran numerous simulations to evaluate the performance of our algorithm and compare it to existing solutions. Experimental results show that our algorithm reaches a complete and stable coverage, within reasonable time and with moderate energy consumption, even when the AoI has an irregular shape, outperforming one of the most acknowledged and cited algorithms [164]. Furthermore, it improves previous approaches producing, when possible, a redundant coverage with guaranteed uniformity.

5.2.4 Algorithm `Push&Pull`

In order to make the exposition clearer, we outline the algorithm, before giving deeper details.

The Idea

Sensors aim at realizing a complete and uniform coverage of the AoI by means of a hexagonal tiling. Notice that the hexagonal tiling corresponds to a triangular lattice arrangement, that is the one that guarantees optimal coverage and density, as discussed in [20], and connectivity, as we detail in Subsection 5.2.6. The algorithm starts with the concurrent creation of several tiling portions. Every sensor not yet involved in the creation of a tiling portion gives start to its own portion in an instant which is randomly selected in a given time interval.

In the following we refer to any starter as $s_{\texttt{init}}$.

The algorithm is based on four main activities, concurrently executed in an interleaved manner. The combination of the described activities expands the tiling and, at the same time, does its best to uniformly distribute redundant sensors over the tiled area, avoiding oscillations.

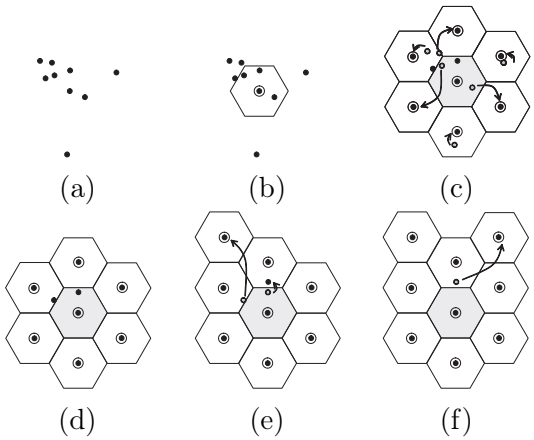

Figure 5.1: Snap and Push example.

Snap activity

The sensor $s_{\texttt{init}}$ elects its position as the center of the first hexagon of its tiling portion. It selects at most six sensors among those located within its *transmitting range* $R_{\texttt{tx}}$ and makes them snap to the center of adjacent hexagons. Newly repositioned sensors, in turn, give start to an analogous selection and snap activity, thus expanding the boundary of the current tiling portion. This process goes on until no other snaps are possible, either because the whole AoI is covered, or because all sensors located at boundary tiles do not have any non-snapped sensor to snap.

Push activity

After the completion of their snapping activity, snapped sensors may still be surrounded by non-snapped sensors located inside their hexagon. In this case, they proactively push such non-snapped sensors towards lower density areas located within their transmitting range. Consequently, non-snapped sensors being in overcrowded areas migrate to low density zones, thus accelerating the coverage process and enhancing its uniformity.

Pull activity

Snapped sensors may detect a coverage hole adjacent to their hexagon and may not find available sensors to snap. In this case, they send hole trigger messages, thus proactively attracting non-snapped sensors in order to make them fill the hole.

Tiling merge activity

The possibility that many sensors act as starters can give rise to several tiling portions with different orientations. In order to characterize and distinguish each tiling portion, the time-stamp of each starter is included in the header of all messages. As a result, messages coming from sensors located in different tiling portions will be characterized by different starter time-stamps. Our algorithm provides a mechanism to merge all these tiling portions into a unique regular and uniformly oriented tiling. Loosely speaking, when the boundaries of two tiling portions come in radio proximity with each other, the one with older starter time-stamp absorbs the other one by making its snapped sensors move into more appropriate snapping positions.

Figure 5.1 shows an example of the execution of the first two activities. Namely, Figure 5.1.a depicts the starting configuration, with nine randomly placed sensors and Figure 5.1.b highlights $s_{\texttt{init}}$ starting the hexagonal tiling. In Figure 5.1.c the starter sensor $s_{\texttt{init}}$ selects six sensors to snap in the adjacent hexagons, according to the minimum distance criterion. Figure 5.1.d shows the configuration after the snap activity of $s_{\texttt{init}}$. In Figure 5.1.e, a deployed sensor starts a new snap activity while $s_{\texttt{init}}$ starts the push activity sending a slave sensor to a lower density hexagon.

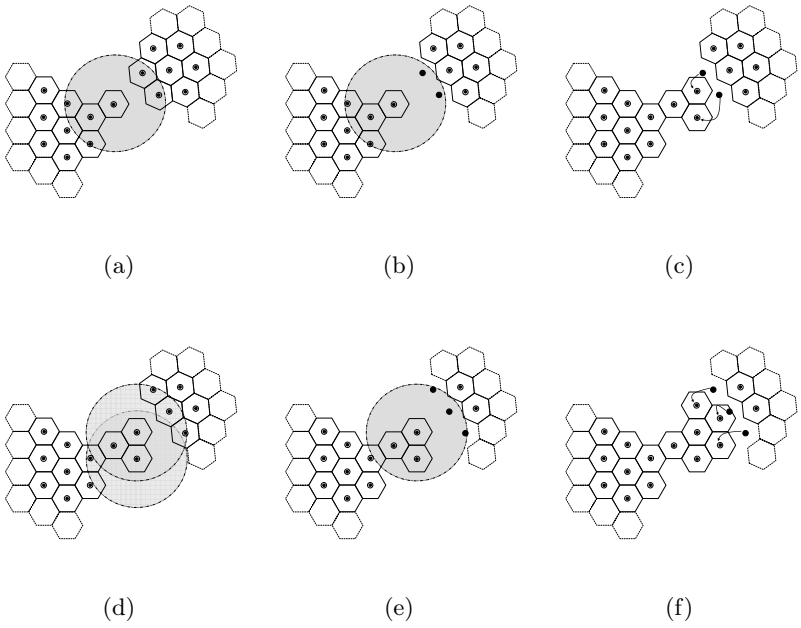

Figure 5.2: Tiling merge activity example.

In Figure 5.1.f the deployed sensor snaps the sensor just received from the starter, reaching the final configuration. Figure 5.2 shows an example of the execution of the tiling merge activity. In particular, Figure 5.2.a shows two tiling portions meeting each other. The portion on the left has the oldest time-stamp, hence it absorbs the other one. Two nodes of the right portion detect the presence of an older tiling and abandon their original tiling (Figure 5.2.b) to honor snap commands coming from a sensor of the left portion (Figure 5.2.c). These just snapped nodes, now belonging to the older portion, detect the presence of three nodes belonging to the right one (Figure 5.2.d) and snap them as soon as they leave their portion (Figures 5.2.e-f).

We defer the introduction of the example regarding the pull activity to the next subsection, when more details will be available to clarify the explanation.

Details of `Push&Pull`

In order to describe the algorithm in more detail, we provide some terminology and notation.

Let V be a set of equally equipped sensors able to determine their own location, endowed with boolean sensing capabilities. We set the hexagon side length $l_{\mathtt{h}}$ to the *sensing radius* $R_{\mathtt{s}}$. This setting guarantees both coverage and connectivity when $R_{\mathtt{tx}} \geq \sqrt{3}R_{\mathtt{s}}$. This requirement is not restrictive as most radio devices can adjust their transmitting range by properly setting their transmitting power.

A sensor being positioned in the center of a hexagon is referred to as a *snapped sensor*. Given a sensor x snapped to the center of a hexagon, we call *slaves* of x all the other sensors lying in its hexagon. All sensors that are neither snapped nor slaves are called *free*. We denote by $S(x)$ the set of slaves of x and by $Hex(x)$ the hexagonal region whose center is covered by x. We define $L(x)$ the set composed by the free sensors located in radio proximity from x and by its slaves $S(x)$.

We now give additional details on the previously sketched activities composing Algorithm `Push&Pull`.

Snap activity

At the beginning of the deployment process, each sensor may act as starter of a snap activity from its initial location at an instant randomly chosen over a given time interval. In order to propagate a tiling portion, a snapped sensor x performs a *neighbor discovery*, that allows x to gather information regarding $S(x)$ and all the free and snapped sensors located in radio proximity from x. As a consequence of the neighbor discovery, x determines whether some adjacent snapping positions are still to be covered and acts as leader of the corresponding snap activity. To give start to new snap activities, x selects the sensor in $L(x)$ which is the closest to each uncovered position and snap it there. A snapped sensor leads the snapping of as many adjacent hexagons as possible. If all the hexagons adjacent to $Hex(x)$ have been covered, x stops any further snapping and gives start to the push activity. If some hexagons are left uncovered but no more sensors in $L(x)$ are available, x starts the pull activity instead.

Push activity

Given two snapped sensors x and y located in radio proximity from each other, x may offer one of its slaves to y and push it inside its hexagon if $|S(x)| \geq |S(y)| + 1$. Notice that, when $|S(x)| = |S(y)| + 1$, the flow of a sensor from $Hex(x)$ to $Hex(y)$ leads to a symmetric situation in which $|S(x)| + 1 = |S(y)|$ possibly causing endless cycles. In such cases we restrict the push activity to only one direction: x pushes its slave to y only if $id(y) < id(x)$, where $id(\cdot)$ is a function initially set to the unique label of the sensor (notice that this is not the only possibility, $id(\cdot)$ could be set for example to a random non negative value). We formalize these observations by defining the following *Moving Condition*, that enables the movement of a sensor from $Hex(x)$ to $Hex(y)$:

$$\{|S(x)| > |S(y)| + 1\} \ \vee \ \{|S(x)| = |S(y)| + 1 \ \wedge \ id(x) > id(y)\}.$$

The snapped sensor x executes a push action by sending one of its slaves s towards the hexagon of a snapped sensor y.

The destination hexagon $Hex(y)$ is selected in a way such that x verifies the moving condition with respect to y. In particular, as a destination of the push action, x selects the closest hexagon among those with the lowest number of slaves. Among the sensors which can be pushed to the destination, x selects the closest to $Hex(y)$.

If a snapped sensor receives a neighbor discovery request while involved in a push activity, it replies as if the ongoing movements were already concluded. Indeed a snapped sensor communicating its own number of slaves without keeping into account the ongoing movements may cause inconsistencies (for example either too many sensors may move to the same hexagon or the same sensor may be offered to several snapped sensors). The snapped sensors involved in a push activity always alert their neighborhood of the changed number of slaves.

Pull activity

The sole snap and push activities are not sufficient to ensure the maximum expansion of the tiling. This may happen when there exists a direction in which the density decreases of at most one sensor, but the Moving Condition is false because of the order relationship induced by function $id(\cdot)$. The same problem may cause also not uniform coverage. For this reason, we introduce the pull activity that makes use of a *trigger mechanism* when some holes occur.

Namely, let x be a snapped node detecting a hole in an adjacent hexagon, with $L(x) = \emptyset$. If x does not have the possibility to receive any slave from its neighbor hexagons, i.e., the Moving Condition is not verified for any of them, then it activates the following trigger mechanism. Sensor x temporarily alters the value of its id function to 0 and notifies its neighbors of this change by mean of a *trigger notification message*. This could be sufficient to make the Moving Condition true with at least a snapped neighbor, so x waits until either a new slave comes into its hexagon or a timeout occurs. If a new slave enters in $Hex(x)$, x sets back its id value and snaps the new sensor, filling

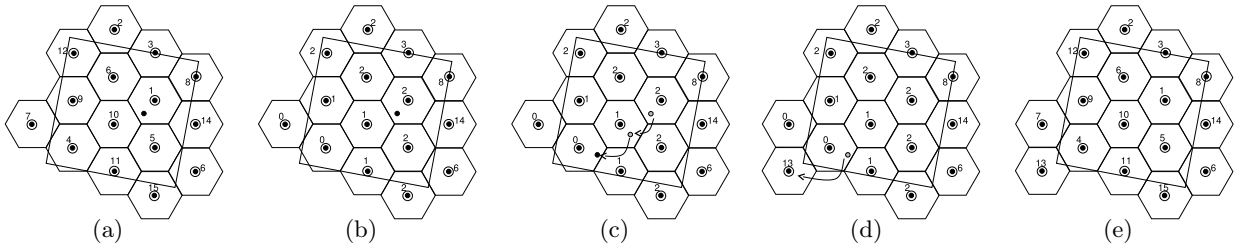

Figure 5.3: Pull activity example.

the hole. If the timeout expires and the hole has not been covered yet, the trigger mechanism is extended through a *trigger extension message* to the adjacent hexagons of x, whose snapped sensors set their id value to 1 and send the related trigger notification message.

This mechanism is iterated by x over snapped sensors at larger and larger distance in the tiling until the hole is covered. Each snapped sensor involved in the trigger extension mechanism sets its id to a value that is proportional to the distance from x. All the timeouts related to each new extension are set proportionally to the maximum distance reached by the trigger mechanism. At this point, as a consequence of timeouts, each involved node sets back its id to the original value.

In order to better detail the trigger mechanism, we show the following example. Figure 5.3.a shows a tiled AoI with a coverage hole in the bottom left corner. Snapped nodes detecting the hole set their id to 0 and send a trigger notification message. As their neighbors do not have slaves, they need to send a trigger extension message, provoking a propagation of the id modification (see Figure 5.3.b). As soon as the unique snapped sensor with a slave sets its id to a new value, the Moving Condition is satisfied and therefore the slave is pushed towards a snapped sensor that is closer to the hole, as shown in Figure 5.3.c. In Figure 5.3.d the hole coverage is highlighted and, after the timeouts expire, all ids are set back to the previous values (Figure 5.3.e).

Notice that more snapped nodes adjacent to the same hole may independently activate the trigger mechanism, possibly at different times. In this case, if a node receives a trigger extension message from two or more nodes, it honors only the one with the lowest id. The detection of several holes may cause the same node y to receive more than one trigger extension message. These messages are stored in a pre-emptive priority queue, giving precedence to the messages related to the closest hole.

Tiling merge activity

The fact that many sensors act as starters implies that several tiling portions with different orientations can be generated. In contrast our algorithm aims at covering the AoI with a perfectly regular tiling thus minimizing overlaps of the sensing disks and enabling a complete and uniform coverage. Hence, we design a merge mechanism according to which as soon as a sensor x receives a neighbor discovery message from another tiling portion it chooses to belong to the oldest one (it discriminates this situation by evaluating the time-stamp of the starter action). Notice that the detection of the sole neighbor discovery messages is sufficient to ignite the tiling merge activity because such messages are sent after any tiling expansion and, if two tiling portions come in radio proximity, at least one of them is increasing its extension. In the following, we call $G_{\texttt{old}}$ and $G_{\texttt{new}}$ the tiling portions with lower and higher time-stamp, respectively. We distinguish three possible cases:

1) x belongs to $G_{\texttt{new}}$:

if x is a slave, it switches its state to free and communicates the new state to the neighborhood. From now on x honors only messages from $G_{\texttt{old}}$ and ignores those from $G_{\texttt{new}}$. This proactive

communication is needed to advertise the presence of $G_{\texttt{new}}$ when there is no message exchange within $G_{\texttt{new}}$ perceivable by the sensors in $G_{\texttt{old}}$. This way, the snapped sensor which x belonged to can properly update its slave set. If x is instead a snapped sensor, it cannot immediately switch its state to free because of its leading role inside $G_{\texttt{new}}$ (e.g. it leads the slave sensors in $S(x)$ and performs push and pull activities). Hence, x temporarily assumes a hybrid role: it advertises itself as free to the nodes of $G_{\texttt{old}}$ and, at the same time, acts as snapped node in $G_{\texttt{new}}$ until it receives a snap command coming from $G_{\texttt{old}}$. After the reception of such a snap command, x moves to the new snap position electing one of its slave as a substitute. If no slave is available, x advertises its departure to its neighbors in $G_{\texttt{new}}$.
2) x belongs to $G_{\texttt{old}}$:

if x is a slave, it ignores all messages from $G_{\texttt{new}}$. If x is snapped, it performs a neighbor discovery, ignores all messages coming from $G_{\texttt{new}}$ (of course apart from the neighbor discovery replies) and honors only messages from $G_{\texttt{old}}$. Observe that the neighbor discovery is necessary to ignite the merge mechanism. The neighbor discovery allows each snapped sensor in $G_{\texttt{old}}$ to collect complete information on nearby sensors that previously belonged to $G_{\texttt{new}}$.
3) x is free:

x honors only messages from $G_{\texttt{old}}$ and ignores those from $G_{\texttt{new}}$.

Balancing Energy Consumption

According to the previous description of `Push&Pull`, slaves consume more energy than snapped sensors, because they are involved in a larger number of message exchanges and movements. We introduce a mechanism to **balance the energy consumption** over the set of available sensors making them exchange their roles. Namely, any time a slave has to make a movement across a hexagon as a consequence of either push or pull activities, it evaluates the opportunity to substitute itself with the snapped sensor of the hexagon it is traversing. As a result, the energy balance is greatly enhanced, though the role exchange has a small cost for both the slave and the snapped sensor involved in the substitution. Indeed, the slave sensor has to reach the center of the current hexagon and perform a *profile packet* exchange with the snapped sensor that has to move towards the destination of the slave. A profile packet contains the key information needed by a sensor to play its new role after a substitution. The criterion at the basis of this mechanism is that two sensors exchange their role whenever the energy imbalance is reduced.

The Sensors Coordination Protocol

The implementation of our algorithm requires the definition of a protocol for the coordination of activities among locally communicating sensors. The coordination protocol provides the rules to solve contentions that may happen in several cases. For example, two or more snapped sensors can decide to issue a snap command to more than one sensor towards the same hexagon tile or a low density hexagon can be selected by several snapped sensors as candidate for receiving redundant slaves. These contentions are solved by properly scheduling actions according to message time-stamps and by advertising related decisions as soon as they are made. This protocol is designed to minimize energy consumption entailing a small number of message exchanges, which is possible because the algorithm decisions are only based on a small amount of local information.

5.2.5 A Discussion on Uniformity Implications

The execution of `Push&Pull` guarantees coverage uniformity only if the number of available sensors is exactly the minimum for the given orientation of the final grid. If redundant sensors are available, their movements are regulated by the moving condition, that impedes the flow of redundant sensors

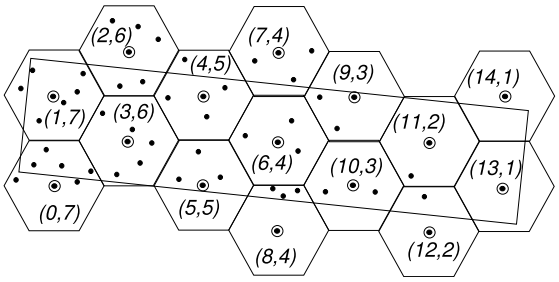

Figure 5.4: Local formation of a stairwise density distribution.

from high density to low density hexagons if the difference between the local densities is only one. This may cause local formation of a stairwise density distribution when the order function is monotonically increasing in the impeded flow direction. An example of such situations is depicted in Figure 5.4, where for each hexagon the elements of the pairs represent respectively the order value and the number of sensors belonging to that hexagon.

The length of these formations is usually very short due to the random distribution of the order value over the set of sensors. The worst case, albeit improbable, happens when a stairwise distribution is as long as the diameter of the AoI.

In order to guarantee the uniformity of the sensor deployment even in presence of redundant sensors, we introduce a *shrinked grid mode* as a variant of the `Push&Pull` algorithm. From now on, we will refer to the basic version with the name PP1 and to the shrinked grid mode with the name PP2. In Section 5.2.6 we prove that PP2 enables a uniformly redundant coverage, and provide metrics and related formulae to calculate the guaranteed redundancy level.

In order to formally describe such mode we introduce the following definition: the *tight number of sensors* is the maximum number of hexagons of side length $l_{\mathtt{h}}$ necessary to cover the AoI for each possible initial position of the sensor set and each possible tiling orientation. We denote this number with $N_{\mathtt{tight}}(l_{\mathtt{h}}, \mathrm{AoI})$, for short $N_{\mathtt{tight}}(l_{\mathtt{h}})$, as the AoI is clear from the context. An upper bound on this number can be calculated by increasing the AoI with a border whose width is the maximal diameter of the tiling hexagon that is $2l_{\mathtt{h}}$ and dividing the area of such a region (call it AoI'($l_{\mathtt{h}}$)) by the hexagon area. Formally:

$$N_{\mathtt{tight}}(l_{\mathtt{h}}) \leq \left\lceil \frac{\mathrm{Area}(\mathrm{AoI}'(l_{\mathtt{h}}))}{(3\sqrt{3}/2)l_{\mathtt{h}}^2} \right\rceil \tag{5.1}$$

Notice that this upper bound is valid in the general case but its calculation can be improved if the AoI has a particularly regular shape.

PP2 is executed with a shorter hexagon side length. Namely, $l_{\mathtt{h}}$ is set to reduce as much as possible the number of slave sensors in the whole deployment, and is calculated as the value that makes the number of sensors equal to the tight number for that side length, and therefore is the inverse function of $N_{\mathtt{tight}}(\cdot)$, calculated in N, where N is the number of sensors. More formally, $l_{\mathtt{h}} = N_{\mathtt{tight}}^{-1}(N)$. Since function $N_{\mathtt{tight}}(\cdot)$ is not known, we calculate an upper bound on $l_{\mathtt{h}}$ as the inverse of the upper bound on $N_{\mathtt{tight}}(l_{\mathtt{h}})$, because $N_{\mathtt{tight}}(\cdot)$ is a decreasing function of $l_{\mathtt{h}}$.

PP1 and PP2 produce sensor deployments with different performance in terms of energy consumption and fault tolerance. The choice between them depends on the particular application requirements, as discussed below here.

In terms of energy consumption, PP2 performs worse than PP1, as we will show and motivate in Section 5.2.7. Nevertheless, as it guarantees a uniformly redundant coverage, it makes possible the use of topology control algorithms [143] that permit selective sensor activation saving energy during the operative phase, which, in turn, follows the deployment phase of the network. Moreover, this

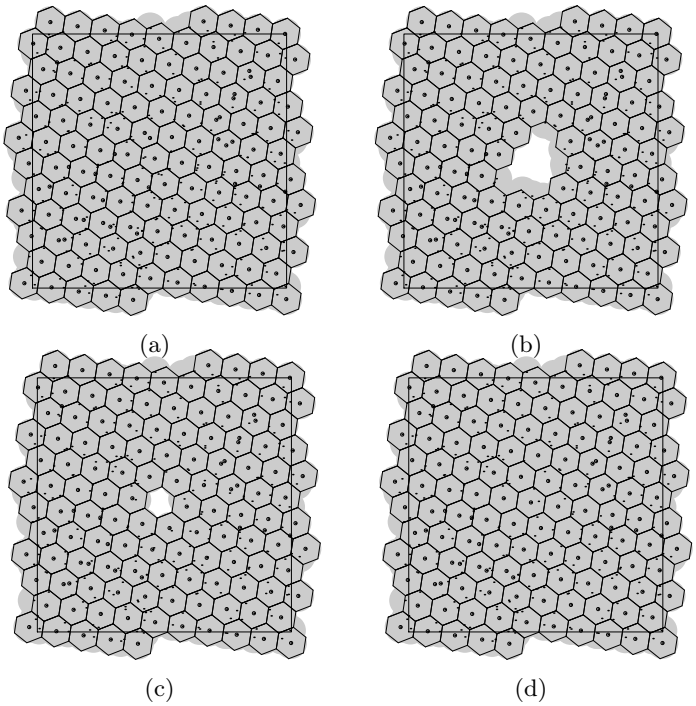

Figure 5.5: Self-healing capability of PP1.

mode is beneficial when the application requires enhanced environmental monitoring and strong fault tolerance capabilities. From the fault tolerance point of view, PP1 may be endowed with a periodic polling scheme to detect new possible coverage holes determined by sensor failures. This way, the detection of new holes causes the restart of the algorithm and the execution of the pull activity that attracts redundant sensors possibly located far from the coverage hole. An example of such a mechanism is shown in Figure 5.5. Overall, PP1 presents self healing properties which are not found in previous solutions.

PP2, instead, can tolerate a number of co-located failures proportional to the redundancy level before leaving a coverage hole but is not able to fill newly detected holes, because (almost) all sensors are snapped and do not take part in the movements determined by the pull activity. For this reason we do not introduce any polling mechanism in PP2, as there are too few slaves available and it would produce an inefficient pull activity in case of a hole detection. On the other hand

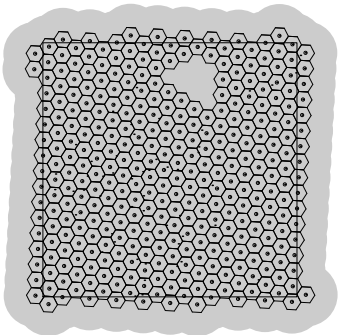

Figure 5.6: Fault tolerance of PP2.

such version guarantees a uniform redundant coverage, if a sufficient number of sensors is availbale, tolerating even numerous sensor failures, as shown in Figure 5.6.

5.2.6 Algorithm Properties

In this section we prove some key properties of the `Push&Pull` algorithm: coverage, connectivity and termination.

Coverage Completeness

In the following we prove that both variants of Algorithm `Push&Pull` guarantee complete sensing coverage of the AoI.

Theorem 5.2.1 *Algorithm* `Push&Pull` *guarantees complete sensing coverage of the AoI, provided that at least the tight number of sensors* $N_{tight}(l_h)$ *are available.*

Proof: Let us assume that a coverage hole exists. As our algorithm is designed, this hole will eventually be detected by a sensor x. Furthermore, by the hypothesis on the number of sensors, it certainly exists a hexagon with at least one redundant slave. Let us call C_x the connected component containing sensor x. Two different cases may occur depending on the position of the redundant slaves with respect to C_x.

If a redundant slave exists in C_x, snapped sensor x starts the trigger mechanism that eventually reaches a redundant slave so that it is pushed towards x and consequently it fills the hole.

If all redundant slaves are located in connected components different from C_x, the area surrounding each connected component is in fact a coverage hole that will eventually be detected by a snapped node located at the boundary. According to the previous case, all the separated connected components containing redundant slaves will expand themselves to fill as many coverage holes as possible. Since, by hypothesis, the number of sensors is at least $N_{\mathtt{tight}}(l_{\mathtt{h}})$, it certainly exists a component containing redundant slaves that will eventually merge in C_x, leading to the situation described in the first case, thus proving the theorem. □

Notice that, having $N_{\mathtt{tight}}(l_{\mathtt{h}})$ sensors is a sufficient condition to guarantee the coverage completeness, but this number is not also necessary. Indeed, $N_{\mathtt{tight}}(l_{\mathtt{h}})$ is calculated as the maximum among all the minimum numbers of sensors necessary to cover the AoI, for each orientation of the final grid with side length $l_{\mathtt{h}}$. So it is possible that $N_{\mathtt{tight}}(l_{\mathtt{h}})$ is larger than the number of sensor strictly necessary for fixed orientation and position of the oldest starter.

Coverage Uniformity

We consider two different coverage redundancy metrics. The first metric evaluates the coverage only in correspondence of the hexagonal grid points. This metric, named *grid coverage level*, is of interest for the applications that do not require a continuous sensing of the area of interest but rely on interpolation of local measurements. On the contrary, the second metric, named *continuous coverage level*, is more restrictive and is introduced to evaluate the coverage redundancy at each point of the area of interest.

Definition 5.2.1 *The* grid coverage level *is the minimum number of sensors covering each point of a regular grid.*

Definition 5.2.2 *The* continuous coverage level *is the minimum number of sensors covering any point of the area of interest.*

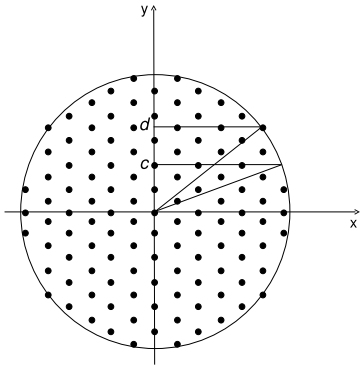

Figure 5.7: Calculus of the grid coverage level.

In order to compute such metrics for PP1 and PP2, we introduce the following lemma.

Lemma 5.2.1 *Given a triangular lattice of side l_h, any circle of radius R and centered in a point of the lattice contains*

$$n(R) = \sum_{i=-\left\lfloor \frac{R}{\sqrt{3}l_h} \right\rfloor}^{\left\lfloor \frac{R}{\sqrt{3}l_h} \right\rfloor} \left(1 + 2 \left\lfloor \frac{\sqrt{R^2 - 3l_h^2 i^2}}{3l_h} \right\rfloor \right) + \; 2 \sum_{i=0}^{\left\lfloor \frac{R}{\sqrt{3}l_h} - \frac{1}{2} \right\rfloor} 2 \left(1 + \left\lfloor \frac{\sqrt{R^2 - 3l_h^2 \left(i + \frac{1}{2}\right)^2}}{3l_h} - \frac{1}{2} \right\rfloor \right)$$

points of the lattice.

Proof: We observe that the points inside the circle of radius R and centered in any point of the lattice always lie in the same position with respect to the center of the circle (see Figure 5.7), then we can slightly modify the reasoning for the well known Gauss' circle problem, dealing with squared grids.

Let the center of the circle be the origin of a Cartesian plane with axis aligned with the grid.

We count the points inside the circle considering them as arranged by horizontal rows.

The number of points in interval $(0, R]$ of x axis is $\left\lfloor \frac{R}{3l_h} \right\rfloor$ and similarly in interval $[-R, 0)$. So, counting the origin, there are $1 + 2 \left\lfloor \frac{R}{3l_h} \right\rfloor$ points in interval $[-R, R]$.

Now we count the number of sensors lying on the rows having a sensor on the y axis.

Let us consider one such row lying on the line $y = c$, it contains $1 + 2 \left\lfloor \frac{\sqrt{R^2 - c^2}}{3l_h} \right\rfloor$ points. As two such consecutive rows in the same semiplane are $\sqrt{3}l_h$ far, it follows that the whole number of sensors on all the rows having a sensor on the y axis and lying on the positive semiplane is

$$\sum_{i=1}^{\left\lfloor \frac{R}{\sqrt{3}l_h} \right\rfloor} \left(1 + 2 \left\lfloor \frac{\sqrt{R^2 - 3l_h^2 i^2}}{3l_h} \right\rfloor \right).$$

Finally, we count the number of sensors lying on the rows not having a sensor on the y axis.

Let us consider one such row lying on the line $y = d$; the sensor closest to the y axis has x-coordinate $\frac{3}{2}l_h$, so we consider the interval $\sqrt{R^2 - d^2} - \frac{3}{2}l_h$ long. Hence, the number of sensors on this row is $2 \left(1 + \left\lfloor \frac{\sqrt{R^2 - d^2} - \frac{3}{2}l_h}{3l_h} \right\rfloor \right)$. With arguments similar to the previous case, we have that the

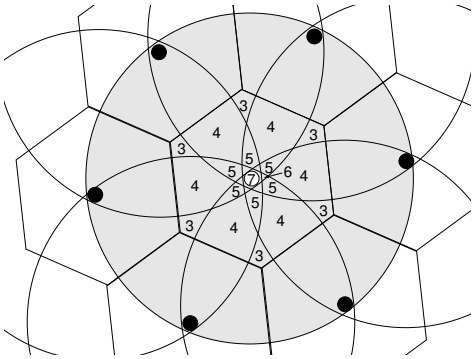

Figure 5.8: Continuous coverage example.

number of sensors lying on these rows in the positive semiplane is:

$$\sum_{i=0}^{\left\lfloor \frac{R}{\sqrt{3}l_{\mathrm{h}}}-\frac{1}{2}\right\rfloor}\left(1+2\left\lfloor\frac{\sqrt{R^2-3l_{\mathrm{h}}^2\left(i+\frac{1}{2}\right)^2}-\frac{3l_{\mathrm{h}}}{2}}{3l_{\mathrm{h}}}\right\rfloor\right).$$

By summing up all the contributions, the result follows. □

Theorem 5.2.2 *Algorithm* `Push&Pull` *guarantees a k grid coverage level, where* $k = n(R_s)$*, provided that at least the tight number of sensors* $N_{tight}(l_h)$ *are available and the shrinked grid mode is enabled.*

Proof: The definition of l_{h} and lemma 5.2.1 imply that, under the given assumptions, algorithm `Push&Pull` provides a complete coverage. Given the geometric regularity of the obtained deployment, every sensing circle surrounding a snapped sensor contains at least a fixed number k of snapped sensors belonging to the triangular lattice determined by the hexagonal grid deployment. As all sensors have the same sensing radius, the sensing redundancy level at the center of the circle is at least k. □

In order to estimate the continuous coverage level of any sensor deployment, in [106] the authors introduce the notion of perimeter coverage. They define a sensor s to be *k-perimeter covered* if all points in the perimeter of the sensing circle of s are covered by at least k sensors (not counting s).

The same authors also prove (see Theorem 1 in [106]) that the sensor deployment provides a continuous coverage level k if and only if each sensor is k-perimeter covered.

Theorem 5.2.3 *Algorithm* `Push&Pull` *guarantees a k continuous coverage level, where* $k \geq \frac{n(R_s)-1}{3} + \frac{n(\sqrt{3}R_s)-n(R_s)}{6}$*, provided that at least the tight number of sensors* $N_{tight}(l_h)$ *are available and the shrinked grid mode is enabled.*

Proof: According to the above cited theorem [106], the level of continuous coverage enabled by the algorithm `Push&Pull` can be calculated as the minimum perimeter coverage over all the snapped sensors. In order to calculate such coverage level, we distinguish two main contributions, the first one coming from sensors located inside the sensing circle of s, and the second one, coming from the sensors located outside.

All sensors located inside the sensing circle of s contribute to the perimeter coverage with a circular sector of amplitude α, with $\frac{2}{3}\pi \leq \alpha < \pi$. Since any of these sensors is symmetric to other five sensors inside the circle, with a rotation of $\pi/3$ centered in the position of sensor s, all the six

of them contribute to at least a double coverage of the sensing circle perimeter of s. The sensors forming this first contribution amount to $n(R_s) - 1$ (not counting the sensor s itself), and all of them globally guarantee $2\left\lfloor \frac{n(R_s)-1}{6} \right\rfloor$-perimeter coverage.

The second contribution is related to the sensors located outside the sensing circle of s. Notice that the sensing circle of sensors located farther than $2R_s$ from s do not intersect the sensing circle of s, while the sensing circle of sensors located at a distance d such that $\sqrt{3}R_s < d \leq 2R_s$ intersect the sensing circle of s, determining a circular arc of amplitude less than $\pi/3$. Since we are calculating a lower bound on the minimum perimeter coverage, we do not consider the contribution of this latter sensors as it does not guarantee a complete perimeter coverage and therefore may not affect its minimum value.

For this reason, as a second contribution to the perimeter coverage, we only consider the sensors located inside the circular crown determined by the radii R_s and $\sqrt{3}R_s$. This sensors contribute to the perimeter coverage with a circular sector of amplitude β, with $\pi/3 \leq \beta < \frac{2}{3}\pi$. Since any of these sensors is symmetric to other five sensors inside the crown, with a rotation of $\pi/3$ centered in the position of sensor s, all the six of them contribute to at least one single coverage of the sensing circle perimeter of s. The sensors forming this second contribution amount to $n(\sqrt{3}R_s) - n(R_s)$ and all of them globally guarantee a $\left\lfloor \frac{n(\sqrt{3}R_s)-n(R_s)}{6} \right\rfloor$-perimeter coverage. Notice that the particular 3-axis symmetry, induced by the hexagonal tiling, make it possible to remove the floor operator from the two terms, as $n(R) - 1$ is always divisible by 6.

By summing up the two contribution to the perimeter coverage, we derive the claimed lower bound. □

Coverage and Connectivity

In this subsection we motivate the choice of the hexagonal tiling and the assumption that the sensors operate with $R_{tx} \geq \sqrt{3}R_s$, that is a less restrictive condition than what is usually required in the literature.

In [168], the authors demonstrate that coverage implies connectivity if and only if R_{tx} is twice the value of R_s. This statement is generally valid regardless of the particular distribution of the sensors over the AoI, whether it is a regular geometrical mesh or a random deployment. A hexagonal tiling with side length R_s is the one that minimizes node density while ensuring coverage completeness at the same time, as argued in [20]. Since our algorithm works exactly with this kind of tiling, which corresponds to a triangular lattice, we can relax the relationship between R_{tx} and R_s. If the sensors are regularly deployed on a hexagonal tiling, the distance between any two tiling neighbors is exactly $\sqrt{3}R_s$, implying the following result.

Theorem 5.2.4 *Under a complete triangular lattice coverage with side length R_s, a necessary and sufficient condition to guarantee connectivity is that $R_{tx} \geq \sqrt{3}R_s$.*

Termination of `Push&Pull`

We conclude this subsection by proving the termination of our algorithm.

Let $L = \{\ell_1, \ell_2, \ldots, \ell_{|L|}\}$ be the set of snapped sensors.

Definition 5.2.3 *A* network state *is a vector* $\mathbf{s}$ *whose i-th component represents the number of sensors deployed inside the hexagon $Hex(i)$ governed by the snapped sensor i. Therefore* $\mathbf{s} =< s_1, s_2, \ldots, s_{|L|} >$ *where* $s_i = |S(i)| + 1$, $\forall i = 1, \ldots, |L|$.

Definition 5.2.4 *A state* $\mathbf{s} = \langle s_1, \ldots, s_{|L|} \rangle$ *is* stable*, if the Moving Condition is false for each couple of snapped sensors in L located in radio proximity from each other.*

Theorem 5.2.5 *Algorithm* `Push&Pull` *terminates in a finite time.*

Proof: As long as new sensors are being snapped, the covered area keeps on growing. This process eventually ends either because the AoI has been completely covered or because the sensors have reached a configuration that does not allow any further expansion of the tiling. Due to Theorem 5.2.1 the latter can only happen when all sensors are snapped and thus the state of the network is stable. In order to prove the theorem, it suffices to prove that, once the AoI is fully covered, the algorithm reaches a stable configuration in a finite time. Therefore we can consider the set of snapped sensors L as fixed. The value of the order function related to each snapped sensor, $id(\ell_i)$, is set during the unfolding of the algorithm, it can be modified only temporarily by the pull activity a finite number of times and remains steady onward. Let us define $f : \mathbb{N}^{|L|} \rightarrow \mathbb{N}^2$ as follows:

$$f(\mathbf{s}) = \left(\sum_{i=1}^{|L|} s_i^2, \sum_{i=1}^{|L|} s_i \cdot id(\ell_i) \right).$$

We say that $f(\mathbf{s}) \succ f(\mathbf{s}')$ if $f(\mathbf{s})$ and $f(\mathbf{s}')$ are in lexicographic order. Observe that function f is lower bounded by the pair $(|L|, \sum_{i=1}^{|L|} id(\ell_i))$, in fact $1 \leq s_i \leq |V|$. Therefore, if we prove that the value of f decreases at every state change, we also prove that no infinite sequence of state changes is possible. To this purpose, let us show that every state change from $\mathbf{s}$ to $\mathbf{s}'$ causes $f(\mathbf{s}) \succ f(\mathbf{s}')$. Let us consider a generic state change which involves the snapped sensors x and y, with x sending a slave sensor to $Hex(y)$. We have that $s_i = s'_i \quad \forall i \neq x, y$, and $s'_x = s_x - 1$ and $s'_y = s_y + 1$. As the transfer of the slave has been done according to the Moving Condition, two cases are possible: either $s_x > s_y + 1$, or $(s_x = s_y + 1) \wedge (id(x) > id(y))$. In the first case, the inequality $s_x > s_y + 1$ implies that $\sum_{i=1}^{|L|} s_i^2 > \sum_{i=1}^{|L|} s'^2_i$. In the second case, since $s_x = s_y + 1$ and $id(x) > id(y)$, lead to $\sum_{i=1}^{|L|} s_i^2 = \sum_{i=1}^{|L|} s'^2_i$ and $\sum_{i=1}^{|L|} s_i \cdot id(\ell_i) > \sum_{i=1}^{|L|} id(\ell_i) s'_i$. Therefore in both cases $f(\mathbf{s}) \succ f(\mathbf{s}')$. The function f is lower bounded and always decreasing by discrete quantities (integer values) at any state change. Thus, after a finite number of steps, it is impossible to perform a further state change, i.e. the network reaches a stable state in a finite time. □

5.2.7 Simulation Results

In order to evaluate the performance of `Push&Pull` and to compare it with previous solutions, we developed a simulator using the wireless module of the OPNET modeler software [140].

We compare our method to one of the most understood and cited algorithms [164], which is based on the construction of the Voronoi diagram determined by the current sensor deployment. According to this approach, each sensor adjusts its position on the basis of a local calculation of its Voronoi cell. This information is used to detect coverage holes and, consequently, calculate new target locations according to three possible variants. Among these variants we chose Minimax, that gives better guarantees in terms of coverage extension. Of this algorithm we adopted all the mechanisms provided to preserve connectivity, to guarantee the algorithm termination, to avoid oscillations and to deal with position clustering [164]. In the rest of this section this algorithm will be named $\text{VOR}_{\texttt{MM}}$.

We set the parameters $R_{\texttt{tx}} = 11$ m and $R_{\texttt{s}} = 5$ m. Such values satisfy the $\text{VOR}_{\texttt{MM}}$ requirement $R_{\texttt{tx}} > 2R_{\texttt{s}}$ detailed in [164] and do not significantly affect the qualitative evaluation of `Push&Pull`. The sensor speed is set to 1 m/sec.

Examples of mobile sensor deployment

We show some examples of deployment evolution under the two `Push&Pull` modes: PP1 and PP2.

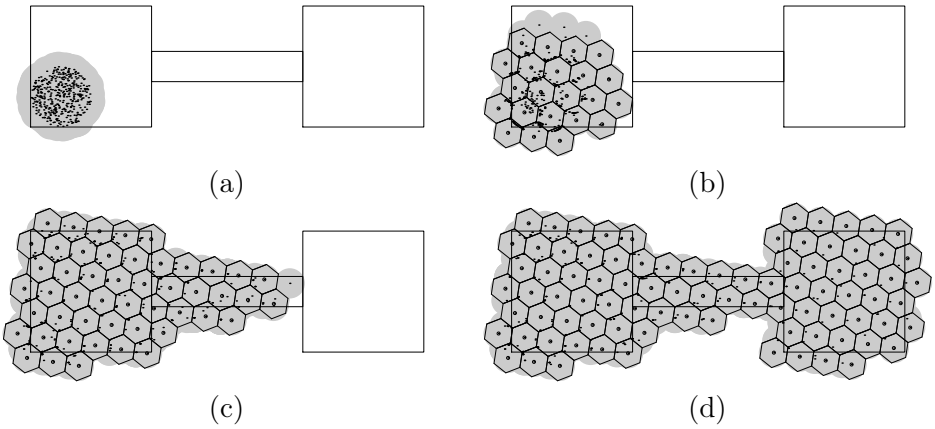

Figure 5.9: Coverage of an irregular AoI under PP1.

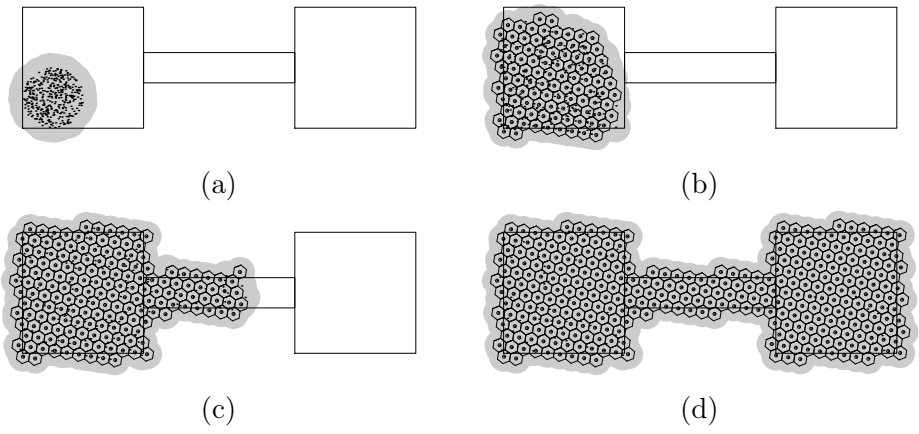

Figure 5.10: Coverage of an irregular AoI under PP2.

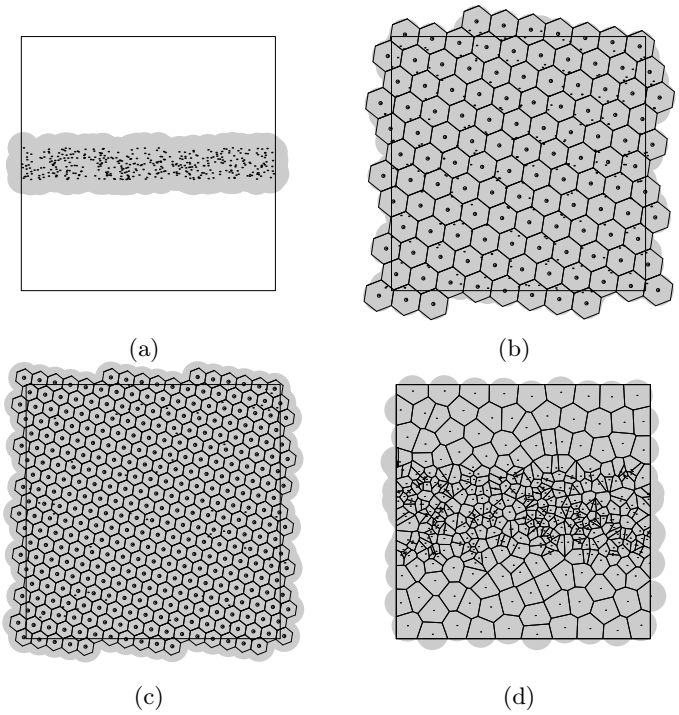

Figure 5.11: a) Trail initial deployment. b) PP1. c) PP2. d) $\text{VOR}_{\texttt{MM}}$.

Figures 5.9 and 5.10 give a synthetic representation of how the sensor deployment evolves under PP1 and PP2 respectively, when 400 sensors are initially located in a high density region. The AoI has a complex shape in which a narrow connects two square regions 40 m $\times$ 40 m. Notice that previous approaches fail when applied to such irregular AoIs. For example, $\text{VOR}_{\texttt{MM}}$ does not contemplate the presence of concavity in the AoI, while the virtual force based approaches are not able to push sensors through narrows [104].

As a second example of sensor deployment, we show experiments conducted with three different starting configurations over an AoI which is a square 80 m $\times$ 80 m. More precisely, in the first experiment, the initial deployment evidences a trail of sensors which crosses the AoI, as shown in Figure 5.11.a. In the second experiment the sensors are densely placed in a corner of the AoI, as shown in Figure 5.12.a. In the third experiment the initial deployment consists in a high density region at the center of the AoI, as shown in Figure 5.13.a.

Notice that the first two initial deployments reflect the realistic scenarios in which sensors are dropped from an aircraft and sent from a safe location at the boundaries of the AoI. The third deployment is introduced as is widely studied in the literature, see for example [164] and [134].

In Figures 5.11, 5.12 and 5.13, the subfigures indicated with b, c and d show the final deployments achieved by PP1, PP2 and $\text{VOR}_{\texttt{MM}}$ respectively.

Performance Comparisons

In the following we compare the performance of PP1, PP2 and $\text{VOR}_{\texttt{MM}}$ when executed over a squared AoI, 80 m $\times$ 80 m.

In order to make reliable performance comparisons, we show the average results of 30 simulation runs (conducted by varying the seed for the generation of the initial deployment).

We compare the behavior of the three algorithms with respect to several performance objectives:

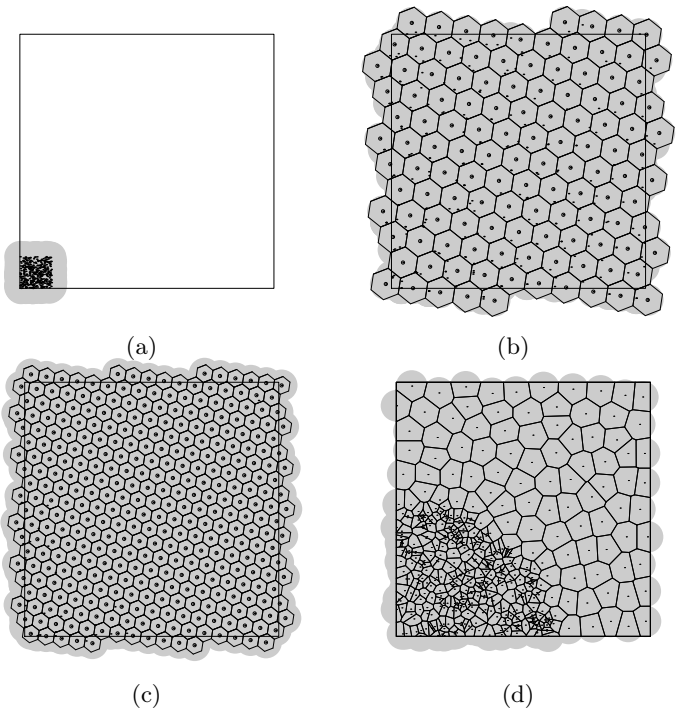

Figure 5.12: a) Safe location initial deployment. b) PP1. c) PP2. d) $\text{VOR}_{\texttt{MM}}$.

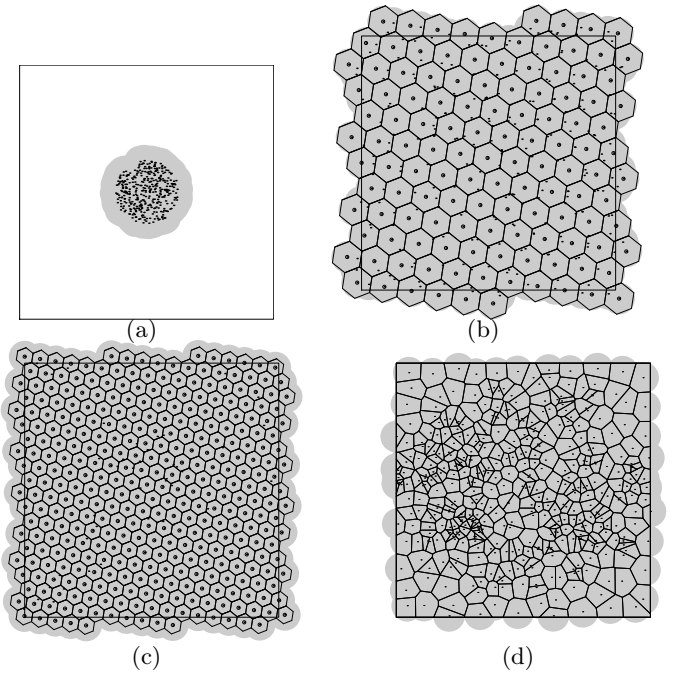

Figure 5.13: a) Central initial deployment. b) PP1. c) PP2. d) $\text{VOR}_{\texttt{MM}}$.

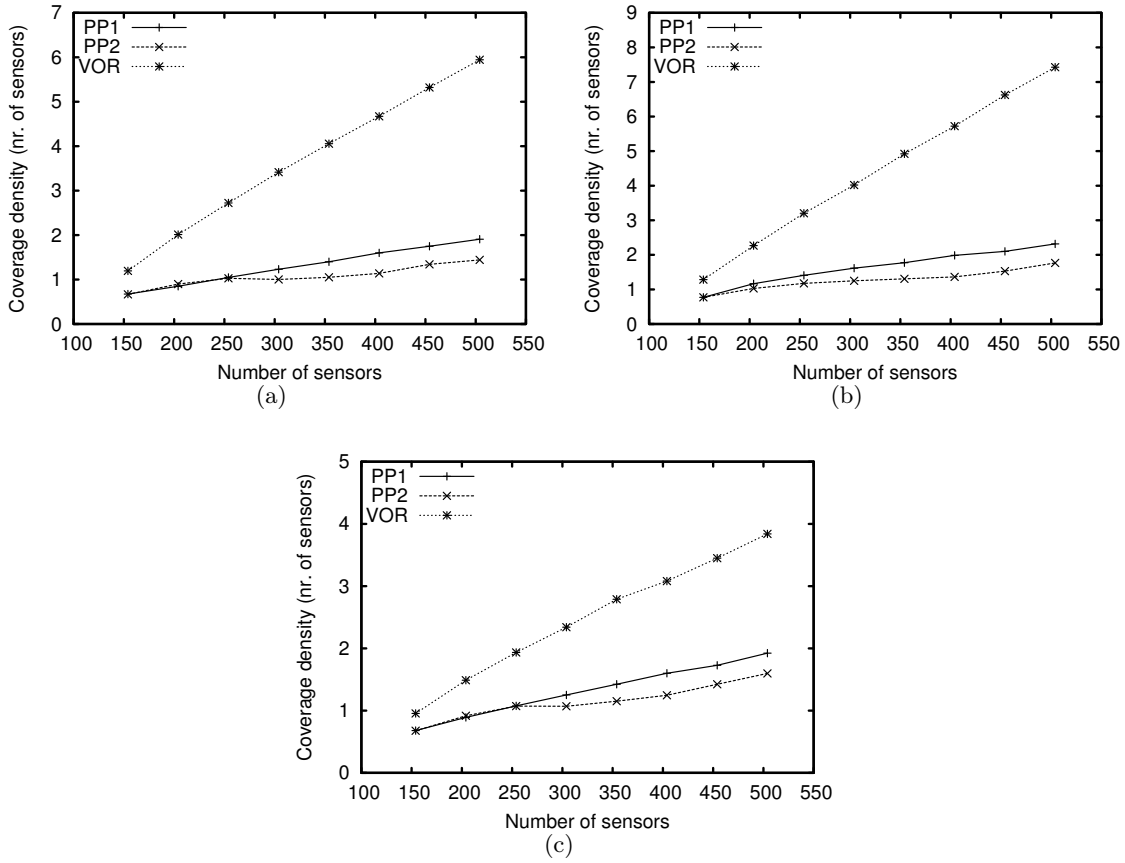

Figure 5.14: Coverage density. Initial deployment: a) trail; b) safe location; c) central.

energy consumption, coverage uniformity, termination and coverage completion time.

All the figures from 5.14 to 5.18 contain three plots each. Plot (a) describes the performance obtained when starting from the trail initial deployment, plot (b) refers to the case in which sensors are initially deployed in a safe corner while plot (c) is related to the case of a dense initial deployment in the center of the AoI. For a better readability, we adopt different scales of the vertical axis for the three scenarios.

Coverage uniformity

The three algorithms give different importance to the uniformity of the coverage. Indeed, `Push&Pull` aims at making the coverage as uniform as possible.

In particular, PP1 builds a coarse grained grid, then it tries to uniform the coverage only on the basis of a local satisfaction of the Moving Condition.

Instead PP2 constructs a fine grained grid by setting the hexagon side at the minimum length which guarantees the full coverage of the AoI, thus making sensors traverse longer distances than other solutions.

On the contrary, $\text{VOR}_{\texttt{MM}}$ aims at covering the AoI regardless of the uniformity of the final coverage, and sensors stop moving when the AoI is fully covered.

In order to evaluate the coverage uniformity, we compute the coverage density as the number of sensors covering the points of a squared mesh with side 1 m.

Figure 5.14 shows the standard deviation of the coverage density. Notice that we do not show the average coverage density because it is not significant, since it only depends on the number of

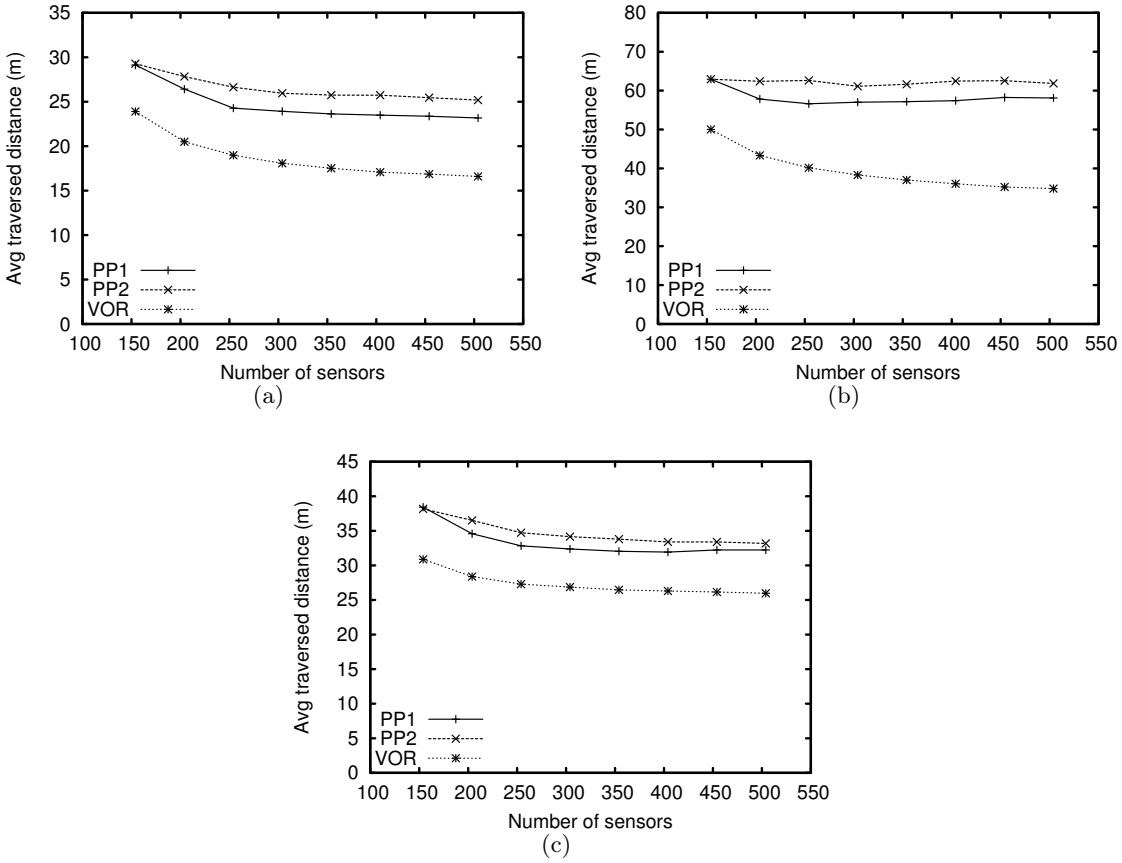

Figure 5.15: Average traversed distance. Initial deployment: a) trail; b) safe location; c) central.

available sensors.

The standard deviation of the coverage density achieved by PP1 and PP2 is smaller than the one obtained by $\text{VOR}_{\texttt{MM}}$. In particular, $\text{VOR}_{\texttt{MM}}$ terminates as soon as the AoI is completely covered, without balancing the density of the sensor deployment, while PP1 and PP2 keep on moving until they balance the coverage.

This result is particularly important as a uniformly redundant sensor placement provides self-healing and fault tolerance capabilities. In the case of PP1, the presence of quite uniformly distributed slaves ensures the self-healing capability of the deployment, while for what concerns PP2, the guaranteed continuous k-coverage gives tolerance up to $k-1$ faults.

Energy consumption

We show an analysis of the energy consumption of the three algorithms in terms of average traversed distance per sensor and average number of starting/braking actions. Finally we give an overall evaluation which also comprises the communication costs.

Average traversed distance per sensor

The different weight that the three algorithms give to the uniformity objective is reflected in the different trends of the average traversed distance shown in Figure 5.15.

The average traversed distance of $\text{VOR}_{\texttt{MM}}$ decreases with the number of sensors. This is due to the fact that more and more sensors maintain their initial positions when no coverage holes are detected. On the contrary, in both modes of `Push&Pull`, all sensors contribute to realize a quite uniform coverage, hence the average traversed distance becomes approximately constant for large

numbers of sensors. This implies that $\text{VOR}_{\texttt{MM}}$ spends less energy in movements than `Push&Pull` at the expense of the uniformity of the final coverage, in all the considered settings of the initial deployment.

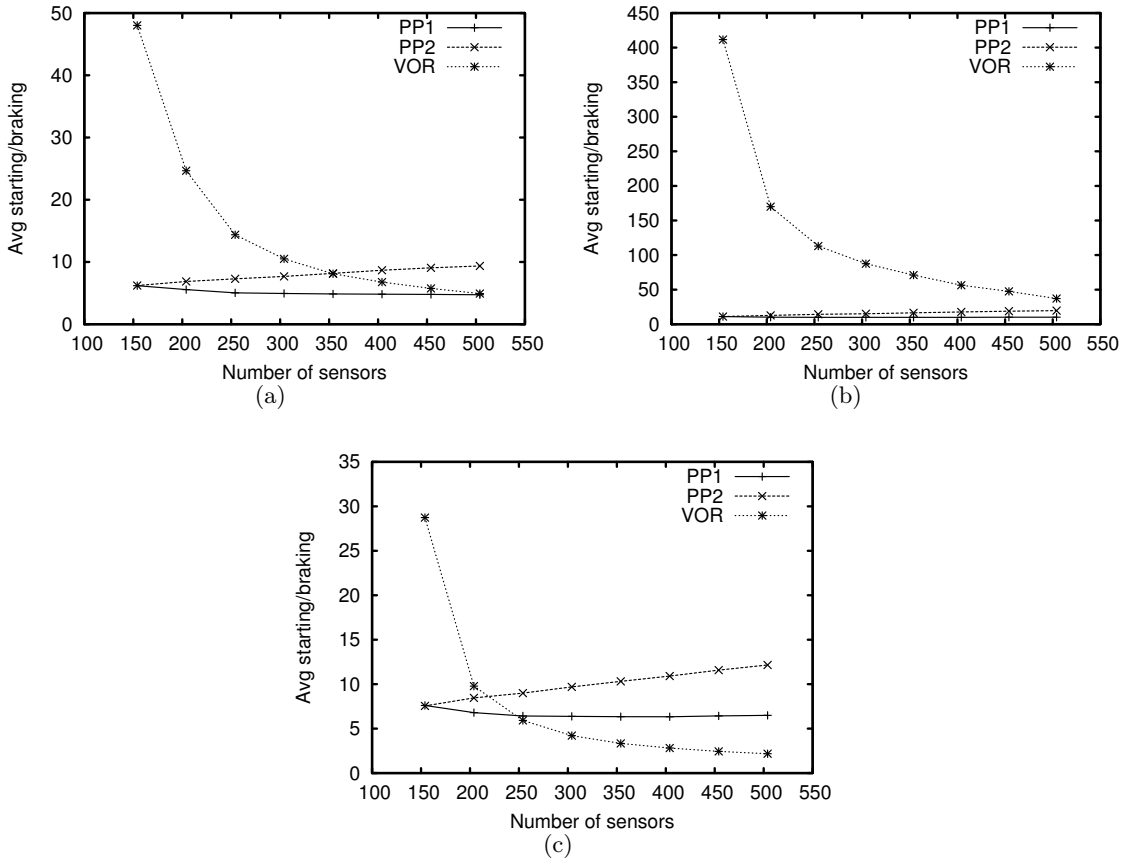

Figure 5.16: Average number of starting/braking. Initial deployment: a) trail; b) safe location; c) central.

Average number of starting/braking actions per sensor

We now consider the number of starting/braking actions as they require a high energy consumption [164]. Figure 5.16 highlights that, when the number of sensors is relatively small, $\text{VOR}_{\texttt{MM}}$ performs a number of starting/braking actions higher than PP1 and PP2. On the contrary, when the number of sensors increases, $\text{VOR}_{\texttt{MM}}$ apparently performs better, showing a rapid decrease of the number of starting/braking actions. This is due to the presence of a growing fraction of sensors which does not move at all, generating a final non uniform coverage as well as a high energy imbalance among sensors. The most critical scenario for the $\text{VOR}_{\texttt{MM}}$ algorithm is the safe location initial deployment (notice the different vertical scales in Figure 5.16).

Average energy consumption

We now analyze the overall energy consumption of the three algorithms. We utilize a unified energy consumption metric obtained as the sum of the contributions given by movements, starting/braking actions and communications. The energy spent by sensors for communications and movements is expressed in energy units. The reception of one message corresponds to one energy unit, a single transmission costs the same as 1.125 receptions [6], a 1 meter movement costs the same as 300 transmissions [164] and a starting/braking action costs the same as 1 meter movement [164].

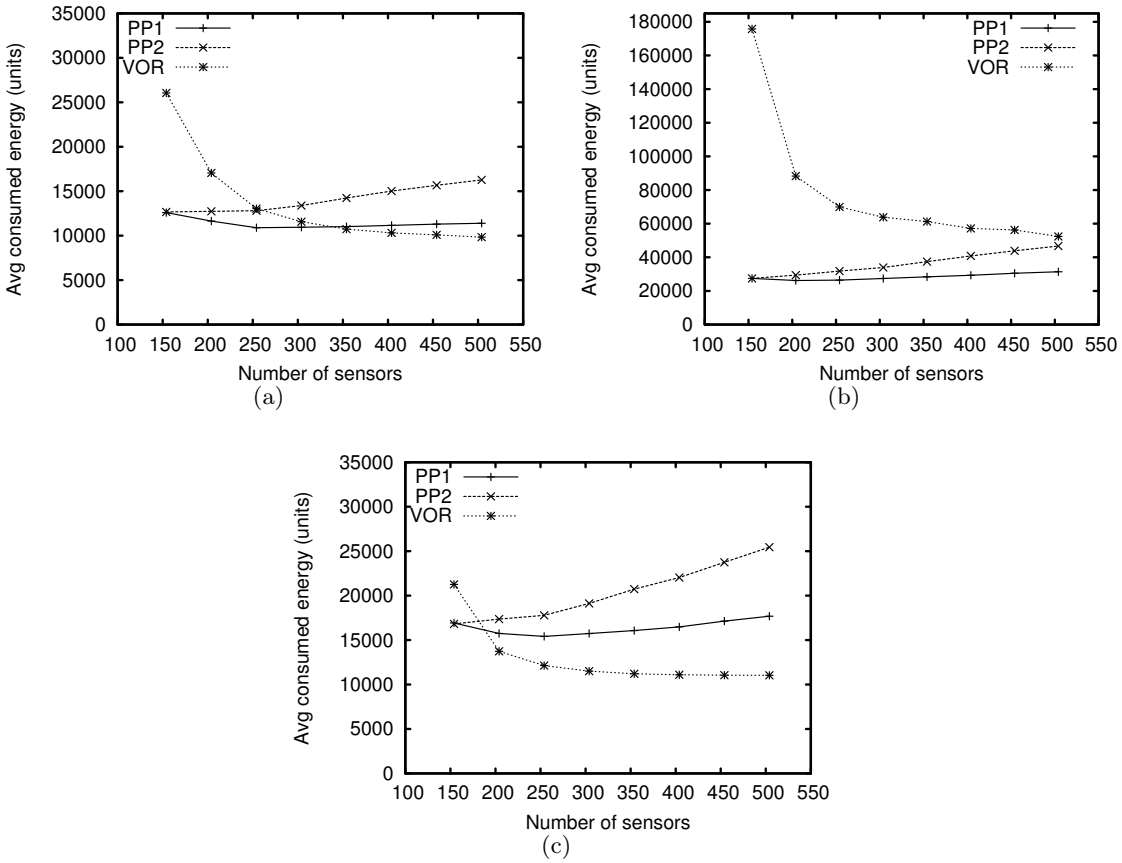

Figure 5.17: Average energy consumption. Initial deployment: a) trail; b) safe location; c) central.

Figure 5.17 shows the energy consumption of PP1, PP2 and VOR_{MM} in the three considered scenarios.

PP1 presents a stable energy consumption even when the number of sensors varies significantly. Indeed, although only a fixed number of them are snapped, all sensors are involved in the push and pull activities, thus improving the coverage density and uniforming the energy consumption.

PP2 instead, shows that the consumed energy increases as the number of sensors grows. Indeed, the more numerous are the sensors, the finer is the grid adopted by PP2. Therefore, in order to reach their destination, the slaves traverse more hexagons, and are involved in a higher number of push activities than in the case of PP1. This increases the number of starting/braking actions as shown in Figure 5.16 and also increases the consumed energy consequently .

VOR_{MM} consumes more energy than PP1 and PP2, when the number of sensors is close to the tight value. Although sensors do not traverse long distances (as shown in Figure 5.15), the limit on the maximum moving distance per round required by VOR_{MM} increases the number of starting/braking actions (see Figure 5.16), thus resulting in a high energy consumption. This effect is particularly evident in the case of the safe location scenario shown in Figure 5.17(b).

The average energy consumption of VOR_{MM} decreases when increasing the number of sensors. Notice that this is not due to a better behavior of the algorithm but to the fact that a greater and greater fraction of sensors do not move at all. This implies that a considerable number of sensors consume a large amount of energy to move from overcrowded regions toward uncovered areas. As soon as all the coverage holes are eliminated, VOR_{MM} stops, leaving some zones with very low density coverage. These zones are prone to the occurrence of coverage holes in case of failures, as the sensor density is very scarce and the only sensors located in proximity have already consumed much energy during the network deployment.

Although PP2 consumes more energy when the number of the available sensors grows, it guarantees a more uniform coverage with respect to VOR_{MM} and PP1. Moreover, the regularity of the final deployment enables the use of topology control algorithms [143] that permit a selective sensor activation, saving energy during the operative phase which follows the deployment.

Coverage completion and termination time

Figure 5.18 shows the coverage and termination time for the three algorithms. Notice that for VOR_{MM} the termination and coverage completion times coincide, while for `Push&Pull` some more movements are still executed even after the coverage completion.

In the three considered scenarios, if the number of sensors available is close to the minimum needed to cover the AoI, VOR_{MM} requires a very long time to complete the coverage, while `Push&Pull` terminates much earlier. When the number of available sensors grows, VOR_{MM} has a shorter termination time, which instead remains stable under PP1. On the contrary, the termination time of PP2 grows when the number of available sensors increases. In particular, VOR_{MM} generally requires more time than PP1 to achieve its final coverage. Only in the case of the central initial deployment, and for a high number of available sensors (N greater than 320) VOR_{MM} terminates in a shorter time if compared with PP2 (see Figure 5.18(c)). This is due to the fact that the termination time of PP2 is delayed by the numerous hole triggers generated by the pull activity.

It is worth noting that as already discussed, the safe location deployment, constitutes a critical scenario for VOR_{MM} as this algorithm works at its best for more uniform initial sensor distributions. Indeed, Figure 5.18(b) shows that VOR_{MM} requires much more time than in the other sets of experiments, (a) and (c), to achieve its final deployment (16000 sec in the case of safe location vs. 1400 sec in the case of trail, and 900 sec in the case of central initial deployment).

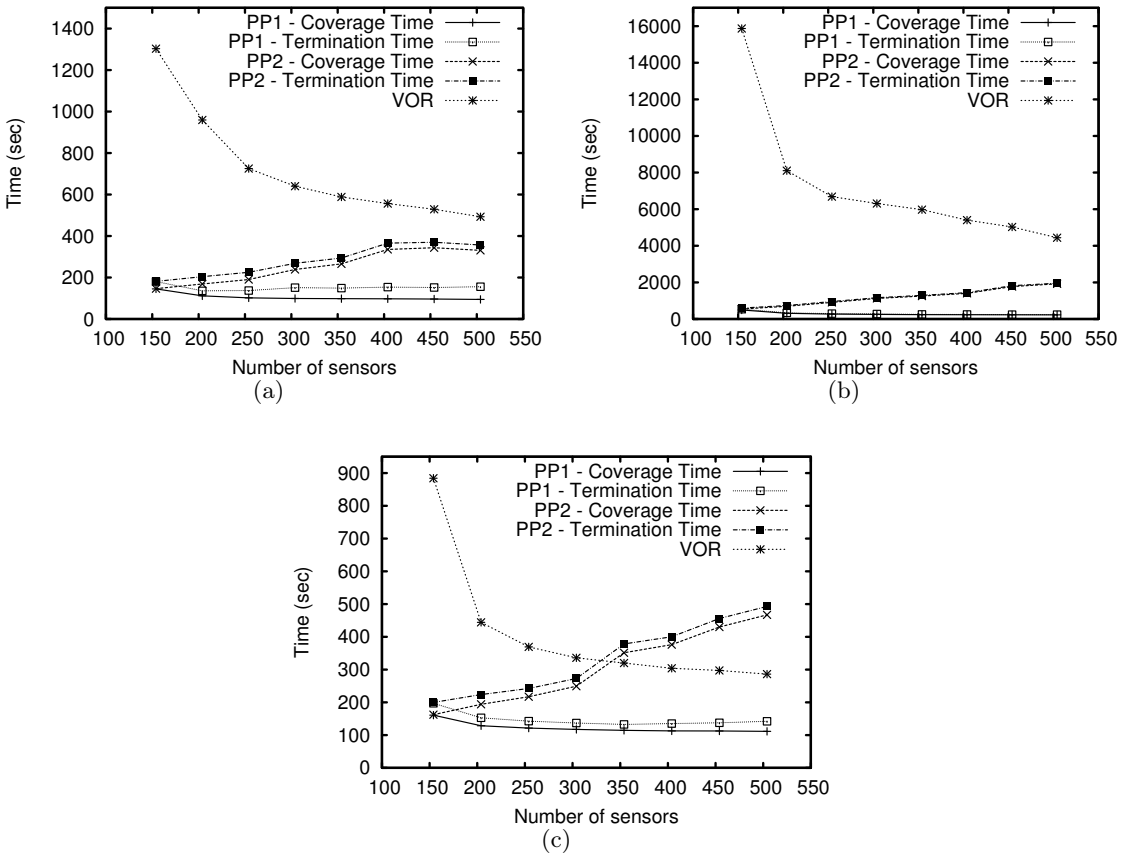

Figure 5.18: Termination and coverage time. Initial deployment: a) trail; b) safe location; c) central.

5.2.8 Conclusion

We proposed an original algorithm for mobile sensor self deployment named `Push&Pull`. According to our proposal, sensors autonomously coordinate their movements in order to achieve a complete and uniform coverage with moderate energy consumption. The execution of `Push&Pull` does not require any prior knowledge of the operating conditions nor any manual tuning of key parameters, as sensors adjust their positions on the basis of locally available information. The proposed algorithm leads to a guaranteed final static and uniform coverage, provided that there is a sufficient number of sensors. As experiments show, `Push&Pull` outperforms previously proposed approaches thanks to its ability to cover target areas of even irregular shape.

Mechanisms for obstacle detection and avoidance are being investigated and considered as future extensions of this work.

5.3 Proxy Assignments for Filling Gaps in Wireless Ad-hoc Lattice Computers

5.3.1 The Model and the Problem

We consider a collection of devices, arranged on a 2-dimensional grid. Each device has computing capabilities and is able to communicate directly with its left, right, top, and bottom neighbor in the grid. Some of the devices can be defective, and thus inable to correctly communicate to some of the neighbors or to perform computation. Defective devices have to be simulated by correctly working devices, acting as proxies.

The proxy assignment we search for has the main objective of minimizing the delay (computed in number of hops) inserted in simulated communication between two adjacent devices, at least one of which is defective.

The motivation that lead us to study this problem comes from the proposal of Gupta et al. [99, 100], of using a collection of small devices capable of radio communication, like PDAs and cell phones, in order to perform simulation of physical phenomena. (The same formulation of the problem, however, applies to parallel computer architectures in which processing units are connected via a 2-dimensional mesh.) Based on the foundations of cellular automata and lattice computers [38,103,135], Gupta et al. [99,100] proposed the use of such an ensemble of radio devices to create a *wireless ad-hoc lattice computer (WAdL)*. The goal of a WAdL is to harness the collective computing capabilities of the devices for the common cause of scientific computing via analogical simulations. The methodology, formalised by earlier work on lattice computers, is (1) to represent, by computing devices logically arranged as a part of a lattice, the region of Euclidean space in which a phenomenon unfolds, and (2) to have the computing devices *analogically* simulate the unfolding of the phenomenon in this representation of Euclidean space.

In analogical simulations on a lattice computer (and hence on a WAdL), the motion of an object across Euclidean space is carried out as a sequence of steps, in time proportional to real time, where in each step the representation of the object may move from one processing element to a *neighbor*, as defined by the underlying lattice of the lattice computer [156]. (See Subection 5.3.3 for a brief discussion of analogical simulations.)

Clearly, the accuracy of the results of these simulations is directly dependent on the resolution of the underlying lattice of a WAdL. Moreover, since the devices in a WAdL are logically, but not necessarily **physically**, at lattice points (in the underlying lattice), the communication distance between devices is not proportional to the physical distance between lattice points represented by those devices. To ensure analogical simulations that unfold in time proportional to the real time unfolding of the phenomenon being simulated, additional adjustments have to be made (see [99,100]) to the lattice computer algorithms described in the literature [38].

Radio devices are moreover prone to errors, as they can move during the execution of the simulation thus loosing connectivity with their logical neighbors, they can run out of battery charge, or simply be turned off. Thus it is necessary to develop self healing mechanisms in order to make WAdLs effective.

5.3.2 Original Contribution

The problem we address in our contribution, is the problem of bridging "gaps", created by faulty devices, in the underlying lattice of a WAdL. We adopt the approach proposed by Moore et al. [138], i.e., to assign, for each faulty device x, an active device, $l(x)$, in the WAdL to serve as a proxy for x.

As a consequence, the communication between two neighboring faulty devices x and y will be carried out as a communication between the two proxy devices $l(x)$ and $l(y)$. These proxy devices,

though, may not be neighbors, and thus analogical simulations being carried out on such a WAdL will run slower than on a WAdL with no faulty devices. Moore et al. [138] presented assignment schemes for square gaps. We extend that work and study gaps of more general shapes. As in [138], our primary goal is to devise assignment schemes for bridging gaps so that the communication time between proxies for any pair of neighboring faulty devices is minimized. We provide lower bounds on the minimum delay caused by gaps with a convex shape, and we develop proxy assignments matching these bounds for lines and rectangles. Our assignments are also optimal with respect to load balance of the proxies.

5.3.3 Analogical Simulations

A physical phenomenon is a development in a region of Euclidean space over a period of time. At each instant in time (in a given time period), the set of objects participating in the phenomenon, together with their attribute values (such as speed, spin, etc.) at that time, completely describes a snapshot in the unfolding of the phenomenon. Most problems in scientific computing are about phenomena whose unfolding involves the motion of participating objects in Euclidean space. Solutions to these phenomena usually involve determining (predicting) the attribute values of objects over time. Some phenomena can be solved analytically using closed form functions of time. On the other hand, there are phenomena where the only apparent method for predicting the attribute values of participating objects at any instant in time, is to **simulate** the unfolding of the phenomenon up through that instant of time [94].

When carried out on a digital computer, such simulations necessarily develop in a discretized representation of a region of Euclidean space, and over discrete time units. Moreover, such simulations must use, at any given instant of simulation time, only information available **locally**, at a discrete point in the represented Euclidean space, to compute the attribute values of participating objects at the next instant of simulation time. Cellular automaton machines [103] and lattice computers [38] provide the necessary framework for a discretized representation of Euclidean space in which to carry out such simulations. Several physical phenomena, including spherical wavefront propagation and fluid flow [77, 167], have been successfully simulated on such a framework where the simulation algorithms do not use the traditional analytical models for the phenomena.

5.3.4 Terminology and Preliminaries

Consider a collection of devices arranged on a subset of the cells of a square grid, one device per cell.

Definition 5.3.1 *A device collection is said to be* row convex *(respectively,* column convex*) if every row (resp. column) of devices forms a contiguous interval of devices within that row (resp. column). The collection is said to be* row-column convex *if it is simultaneously row convex and column convex.*

The notion of row-column convex collections is similar to the notion of hv-convex polyominoes discussed in the literature [47, 124].

As discussed in earlier work on WAdLs [100], the expectation is to harness the collective power of mobile devices in a given geographic region, say a few blocks in the downtown area of a city. It is quite likely that some buildings in that area do not have wireless coverage, or that the office occupying some building is closed on a particular day, and thus there are no mobile devices in the physical space occupied by that office. Such situations leads to gaps in a WAdL in that area, and such gaps can be faithfully modeled by a row-column convex collection of devices. Indeed, here we assume that the collection of faulty devices in a WAdL is row-column convex. We refer to such a collection of faulty devices in a WAdL as a *gap* or a *hole* in the WAdL.

Thus the overall picture is of a row-column convex device collection consisting of active devices surrounding a row-column convex gap of defective devices. The set of active devices that are immediately adjacent to the defective ones form a contiguous contour that we call the *active perimeter* of the gap.

As mentioned in Section 5.3.1, to simulate lattice computations completely, we now seek to assign to every defective device, a corresponding device on the active perimeter that will serve as its *proxy* in the computation; for convenience, we think of active perimeter devices as being trivially their own proxies. Consider a device x and its proxy $l(x)$. If device x were active, it would have been able to communicate directly with adjacent devices along its row and its column. However, if device x is defective, communication for x is now handled instead by the proxy $l(x)$. Additionally, communication takes place along paths whose individual hops (edges) only connect active devices.

Accordingly, in the presence of a hole, the distance in hops, $d(x, y)$, between any two active devices at positions x and y in the square grid is the length of the shortest path between x and y that is composed entirely of horizontal hops (along a row) or vertical hops (along a column) between adjacent active devices. In general, we expect that the communication between two proxies $l(x)$ and $l(y)$ takes place along the shortest path on the active perimeter between $l(x)$ and $l(y)$. Indeed, if the active perimeter encloses a row-column convex gap, then the shortest distance between any two active devices x and y on the active perimeter equals the length of the shortest path that only uses the perimeter edges.

Definition 5.3.2 *For any proxy assignment that maps a device x to a proxy $l(x)$ on the active perimeter, we define the* dilation, load *and* congestion *of the assignment as follows:*

Dilation: *The maximum distance in hops, $d(l(x), l(y))$, taken over all pairs of adjacent devices x and y. (Notice that, in the literature, the notion of dilation is not restricted to neighboring devices.)*

Load: *The maximum number of devices assigned to any active perimeter device.*

Congestion: *The maximum, over every active perimeter edge e, of the number of distinct proxy-communication paths that simulate one-hop communications between two devices, at least one of which is defective.*

Henceforth, for convenience, we use *proxy assignment* to mean "proxy assignment of defective devices to the active perimeter devices". Our problem can be stated as follows: for any row-column convex collection, $\mathcal{C}$, of defective devices in a WAdL, find a proxy assignment such that the dilation, load and congestion of the assignment are minimized. If the dilation of a proxy assignment is the best possible we call that assignment a *dilation-optimal* assignment (similarly, *load-optimal* assignment and *congestion-optimal* assignment).

Definition 5.3.3 *For any row-column convex collection $\mathcal{C}$, $R(\mathcal{C})$ is the unique, smallest rectangle of cells such that $R(\mathcal{C})$ contains $\mathcal{C}$ and each border row and column of $R(\mathcal{C})$ contains at least one element of $\mathcal{C}$. $r(\mathcal{C})$ and $c(\mathcal{C})$ denote the number of rows and columns, respectively, of $R(\mathcal{C})$, containing $\mathcal{C}$. $p(\mathcal{C})$ denotes the number of active perimeter devices around $\mathcal{C}$.*

When the context is clear, we refer to the above quantities as simply r, c and p. Without loss of generality, we assume that the top left corner cell of $R(\mathcal{C})$ has coordinates $(1, 1)$, and the bottom right corner cell of $R(\mathcal{C})$ has coordinates (r, c). We refer to the devices by the coordinates of the cell that the device occupies.

Figure 5.19 illustrates the above definition. Figure 5.19 also illustrates our reference coordinate system for naming devices. The active perimeter devices are shown shaded.

It is, then, easy to verify the following two lemmas.

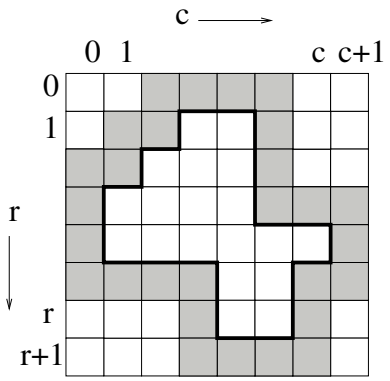

Figure 5.19: A row-column convex gap. $r = 6$, $c = 6$, $p = 28$.

0	1	2	3					c	c+1
2c+5	f_1	f_2						f_c	c+2
2c+4	2c+3							c+4	c+3

Figure 5.20: Numbering of devices for a single-row gap.

Lemma 5.3.1 *For any row-convex collection $\mathcal{C}$,*

$$p(\mathcal{C}) = p(R(\mathcal{C})) = 2(r(\mathcal{C}) + c(\mathcal{C})) + 4.$$

Lemma 5.3.2 *For any gap, the load of any proxy assignment is at least the ratio of the number of defective devices to the number of active perimeter devices.*

5.3.5 Single Row Gaps

In this subsection we deal with the special case of connected gaps where $r = 1$. Clearly, such gaps are row-column convex. We first establish a lower bound on the dilation of any proxy assignment for such a gap. Then we discuss a dilation-optimal and load-optimal assignment scheme for single-row gaps.

Lower Bound on Dilation

If a gap contains a single row that has one defective device, i.e., $r = c = 1$, then it is easy to verify that the dilation of any proxy assignment must be at least 3. The following theorem establishes a lower bound on the dilation of proxy assignments for single-row gaps with more than one defective device.

Theorem 5.3.1 *In a WAdL containing a collection $\mathcal{C}$ of defective devices such that $r(\mathcal{C}) = r = 1$ and $c(\mathcal{C}) = c \geq 2$, every proxy assignment of active perimeter devices to the defective ones has dilation at least $D = \lceil (2c + 5)/3 \rceil$.*

Proof: Given a collection $\mathcal{C}$ of defective devices such that $r(\mathcal{C}) = r = 1$ and $c(\mathcal{C}) = c \geq 2$, the active perimeter $p(\mathcal{C}) = p = 2c + 6$. We denote the defective devices as $f_1, f_2, \ldots, f_c$ proceeding from left to right and the active perimeter devices as $0, 1, \ldots, 2c + 5$ starting from cell $(0, 0)$ and proceding clockwise (see fig 5.20 for an example). For convenience, for every $x \in [1, c]$, we denote the devices

$2c+4-x$ as $b(x)$; with this notation, the defective device f_x has device x as its neighbor above and device $b(x)$ as its neighbor below. Thus, in any proxy assignment l that assigns active device $l(x)$ to the defective device f_x, the dilation is lower bounded by the maximum taken over the following four sets of distances:

1. $\{d(l(x), x) \mid x \in [1, c]\}$,
2. $\{d(l(x), b(x)) \mid x \in [1, c]\}$,
3. $\{d(l(x), l(x+1)) \mid x \in [1, c-1]\}$,
4. $\{d(2c+5, l(1)), d(l(c), c+2)\}$.

Let $D = \lceil (2c+5)/3 \rceil$ (about a third of the active perimeter), and assume, by way of contradiction, that L has dilation at most $D-1$. Let $A(x)$ denote the possible range of values that $l(x)$ can take if $d(l(x), x)$ and $d(l(x), b(x))$ both respect the assumed dilation bound, *i.e.* if both these distances are not more than $D-1$. Consider the specific defective devices f_x and f_y where x and y are given by:

$$y = D - 1 = \lceil (2c+5)/3 \rceil - 1$$

and

$$x = (c+1) - y = \lfloor (c+1)/3 \rfloor.$$

Note that x and y are equi-distant from the left and the right ends of the defective row. From the definition of y, it is easy to see that the distance to $b(y)$ from any device in the range $[0, y-1]$ is greater than D. Similarly, the distance to y from any device in the range $[b(y)+1, 2c+5]$ is at least D. Consequently, $A(y)$, the allowable range for $l(y)$, is the interval $[y, b(y)]$ that consists of the devices to the right of (and including) y (respectively, $b(y)$) in the row above (respectively, row below) the defective row. Similar reasoning yields the complementary fact that $A(x)$, the admissible range for $l(x)$, is the interval $[b(x), x]$ (in the circular ordering in clockwise order around the perimeter). Note that $A(x)$ contains exactly those devices to the left of (and including) x (respectively, $b(x)$) in the row above (respectively, row below) the defective row.

In fact, a more general characterization can be given for $A(x+i)$ as i ranges from 0 through $(y-x)$: $A(x+i)$ consists of two disjoint intervals (not both empty) containing the left and the right ends of the perimeter. Specifically, the proxy $l(x+i)$ either lies in the interval $A_l(x+i) = \{2c+4-(x-i), \ldots, x-i\}$ (in circular clockwise order) or in the interval $A_r(x+i) = \{c+3-i, \ldots, c+1+i\}$; we have

$$A(x+i) = A_l(x+i) \bigcup A_r(x+i)$$

for all $i \in [0, (y-x)]$. For the extreme values of i in its range, we have $A(x) = A_l(x)$ with $A_r(x)$ empty, and have $A(y) = A_r(y)$ with $A_l(y)$ empty. So as one considers the proxies proceeding from f_x towards f_y, there must come a point where $l(x+i)$ is from the left admissible interval $A_l(x+i)$ but the next proxy, $l(x+i+1)$, is from the right admissible interval $A_r(x+i+1)$, *i.e.* the proxies are from opposite sides of the perimeter.

But we require $d(l(x+i), l(x+i+1))$ to be at most the assumed dilation. However, the shortest perimeter distance between any such pair of proxies is no less than the distance between the closest pair of devices from $A_l(x+i)$ and $A_r(x+i+1)$ respectively, *viz.* the distance between $x-i \in A_l(x+i)$ and $c+3-i \in A_r(x+i+1)$. In summary,

$$\begin{aligned} d(l(x+i), l(x+i+1)) &\geq d(x-i, c+3-i) \\ &= c+3-x \\ &= D+1. \end{aligned}$$

This contradicts the assumed dilation bound of $D-1$, and yields the result. □

Note that the same argument also works for a gap consisting of a single column, and, with minor modifications, for any row-convex gap consisting of connected single rows and columns such that each device in the gap has no more than two neighbors in the gap.

Dilation-Optimal Assignment

The following theorem states that the lower bound for dilation for a gap of one row and $c \geq 2$ columns given in Section 5.3.5 is tight. The proof is constructive as it provides a dilation-optimal and load-optimal proxy assignment.

Theorem 5.3.2 *There exists a dilation-optimal and load-optimal proxy assignment for a gap with one row and $c \geq 2$ columns.*

Proof: Let $d = \lceil (2c+5)/3 \rceil$ and $x = \lfloor \frac{c}{2} \rfloor + 1 + d$.
Assign $l(\lfloor \frac{c}{2} \rfloor + 1) = x$, and $l(\lfloor \frac{c}{2} \rfloor) = x + d$.
Complete the proxy assignment l as follows:

- $l(\lfloor \frac{c}{2} \rfloor + i + 1) = x - i$ for $1 \leq i \leq \lceil \frac{c}{2} \rceil - 1$;
- $l(\lfloor \frac{c}{2} \rfloor - j) = (x + d + j) \pmod{2c+6}$ for $1 \leq j \leq \lfloor \frac{c}{2} \rfloor - 1$.

It is easy to see that l is feasible and load-optimal, and its dilation is equal to $\lceil (2c+5)/3 \rceil$.

From this result and Theorem 5.3.1, the claim follows. □

5.3.6 Row-Column Convex Gaps

In this subsection we first establish a lower bound on the dilation of proxy assignments for general row-column convex gaps with $r, c > 1$. Specifically, we show that every proxy assignment has dilation greater than or equal to about one-fourth of the active perimeter (rather than about one-third for the case discussed in the case of single-row gaps (see Theorem 5.3.1 above)).

Then we restrict our attention to rectangular gaps with $r, c > 1$ and present an algorithm for a dilation-optimal and load-optimal proxy assignment for such gaps.

Lower Bound on Dilation

Theorem 5.3.3 establishes a lower bound on the dilation of any proxy assignment for row-column convex gaps. The idea of the proof for the theorem is inspired by the proof of Sperner's lemma in Aigner et al. (see [3], page 148).

Theorem 5.3.3 *In a WAdL containing a row-column convex hole with $r, c > 1$ and active perimeter p, every proxy assignment of active perimeter devices to the defective ones in the hole has dilation at least $D = \lceil p/4 \rceil$.*

Proof: Any row-column convex hole of defective devices can be easily shown to have an even number of devices on its active perimeter. Hence, consider a row-column convex hole with active perimeter p. Starting at an arbitrary position on the perimeter and proceeding clockwise, we *color* the active perimeter devices as follows:

- The first $\lceil p/4 \rceil$ devices are colored A.
- The next $\lceil p/4 \rceil$ devices in order are colored B.
- The next $\lfloor p/4 \rfloor$ devices in order are colored C.

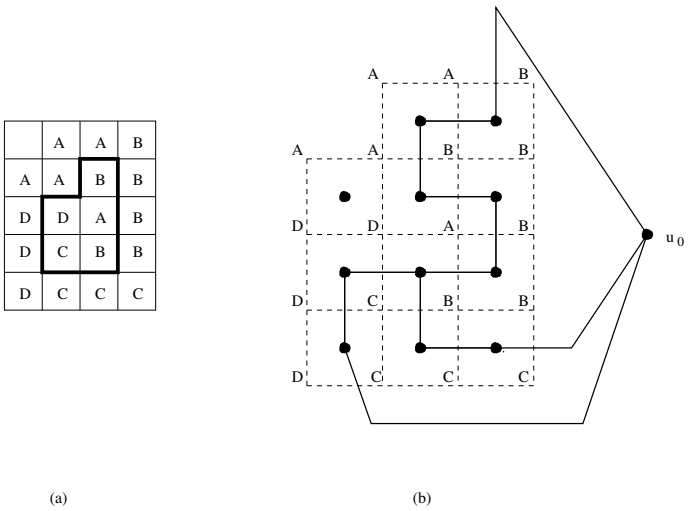

Figure 5.21: a) Coloring induced by a proxy assignment. b) Dual graph.

- The last $\lfloor p/4 \rfloor$ devices are colored D.

Consider any proxy assignment l that assigns an active perimeter device $l(x)$ to a defective device x within the gap. As usual, this proxy device is now responsible for simulating the communication that would have ordinarily been initiated by the device x, now defunct. The proxy assignment induces - in a natural way - a coloring of the defective devices: the device x is given the same color (A, B, C or D) as that given to its active proxy $l(x)$.

Given assignment l, let G_l be the corresponding induced subgraph of the mesh consisting of cells corresponding to the active perimeter devices and the defective devices in the row-column convex gap enclosed by the active perimeter. The edges of G_l replicate the mesh connectivity among adjacent devices and each vertex of G_l is assigned the same color (A, B, C or D) as its corresponding device.

Clearly, G_l is a planar graph with square faces, except for the outer face that traces the active perimeter contour. We now define a special kind of *planar dual* of G_l as follows. Every face f of G_l (including the outer face) corresponds to a unique vertex u_f in the dual graph G_l^d. However, unlike the standard planar dual, vertices u_f and $u_{f'}$ are connected by an edge in G_l^d iff f and f' are adjacent faces and *the shared edge between the adjacent faces f and f' has its endpoints colored AB or BC or CD.* (Note that G_l is not the standard planar dual.)

Figure 5.21(a) shows a row-column convex gap with $r = 3$ and $c = 2$, and the coloring induced by a proxy assignment. Note that $p = 14$, and so there are 4 active perimeter devices colored A, 4 colored B, 3 colored C and 3 colored D. Figure 5.21(b) shows the connectivity graph of the devices in dotted lines and the dual graph as defined above in bold lines. The dual graph vertex corresponding to the outer face is labelled u_0.

Consider the dual vertex u_0 that corresponds to the outer (perimeter) face of G_l. Since the perimeter is colored in sequence with contiguous As, Bs, Cs and Ds, it follows that u_0 has degree 3 witnessed by the unique edges colored AB, BC and CD at the color transitions. All other dual vertices correspond to square faces of G_l and hence have degree at most 4. Moreover, by the handshake lemma for graphs, it follows that among the remaining dual vertices (excluding the odd-degree vertex u_0), there *must be an odd number of vertices of odd degree.* In particular, there must be at least one dual vertex u_f that corresponds to a mesh square (face) f and has degree 1 or 3.

It is easy to verify that up to rotational and mirror symmetries, a dual vertex u_f has degree 3 if and only if the four corners of face f in G_l are colored $ABCD$ in cyclic order. Also, a dual vertex u_f has degree 1 if and only if face f in G_l has corners colored $ABBD$, $ABDA$, $ABDD$, $BCAD$, $CDAC$ or $CDDA$ in cyclic order (modulo rotations and mirror symmetries).

However, recall that the colors identify groups of active perimeter devices that serve as proxies for defective devices in the gap. Hence, any edge of G_l whose endpoints are colored AC or BD corresponds to a pair of proxies that must be at least $\lceil p/4 \rceil$ apart along the perimeter. Since the shortest distance between any two perimeter devices is indeed along the perimeter, we immediately see from the preceding paragraph that all the faces f whose dual vertices u_f have degree 1 witness a dilation of at least $\lceil p/4 \rceil$ for the assignment l.

This leaves the only remaining possibility of a face f with corners colored $ABCD$ in cyclic order. Noting that the corners correspond to four proxies $\bar{a}, \bar{b}, \bar{c}$ and $\bar{d}$ colored A, B, C and D, respectively, along the perimeter. Again, noting that the shortest distance between proxies is along the perimeter, we have

$$d(\bar{a}, \bar{b}) + d(\bar{b}, \bar{c}) + d(\bar{c}, \bar{d}) + d(\bar{d}, \bar{a}) \quad = \quad p,$$

and hence, at least one of the distances must be greater than or equal to $\lceil p/4 \rceil$. □

Dilation and Load Optimal Assignment for Rectangular Gaps

Here we provide a constant time algorithm for a dilation-optimal proxy assignment when the gap is rectangular with at least 2 rows and 2 columns. Our proxy assignment is load-optimal as well when the number of rows and the number of columns are even.

Theorem 5.3.4 *There exists a dilation-optimal proxy assignment for rectangular gaps with $r, c > 1$. Moreover, this proxy assignment is load-optimal when r and c are even.*

Proof: The assignment scheme when $r > 1$ must take into account the parity of both r and c. We give three different schemes for the following three cases:

1. r and c are even;
2. $r + c$ is odd;
3. r and c are odd.

Recall that for each $0 \leq a \leq r + 1$ and $0 \leq b \leq c + 1$, (a, b) denotes the devices in the a-th row and b-th column. Of these, the devices (a, b) where $1 \leq a \leq r$ and $1 \leq b \leq c$ are the defective devices. We number the $p = 2(r + c) + 4$ active perimeter devices sequentially from 0 starting with the device $(0, 0)$ and proceeding clockwise (see Figure 5.22 for an example).

Case 1:
Compute $d = \frac{c+r}{2} + 1$.
For $1 \leq i \leq \frac{c}{2}$ assign:

- defective device $\left(\frac{r}{2}, i\right)$ to active perimeter device i for $1 \leq i \leq \frac{c}{2}$;
- defective device $\left(\frac{r}{2}, \frac{c}{2} + i\right)$ to active perimeter device $\frac{c}{2} + d + 1 - i$;
- defective device $\left(\frac{r}{2} + 1, \frac{c}{2} + i\right)$ to active perimeter device $\frac{c}{2} + 2d + 1 - i$;
- defective device $\left(\frac{r}{2} + 1, i\right)$ to active perimeter device $3d + i$.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
47							46	10							16
46				A			45	9				B			17
45							44	8							18
44	1	2	3	4	5	6	7	19	18	17	16	15	14	13	19
43	37	38	39	40	41	42	43	31	30	29	28	27	26	25	20
42							32	20							21
41				D			33	21				C			22
40							34	22							23
39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24

Figure 5.22: Assignment scheme applied to a rectangular gap of 8 rows and 14 columns.

For $1 \leq j \leq \frac{r}{2} - 1$ assign:

- defective device $\left(\frac{r}{2} - j, \frac{c}{2}\right)$ to active perimeter device $2c + \frac{3}{2}r + 3 + j$;
- defective device $\left(\frac{r}{2} - j, \frac{c}{2} + 1\right)$ to active perimeter device $\frac{c}{2} + j$;
- defective device $\left(\frac{r}{2} + 1 + j, \frac{c}{2} + 1\right)$ to active perimeter device $c + \frac{r}{2} + 1 + j$;
- defective device $\left(\frac{r}{2} + 1 + j, \frac{c}{2}\right)$ to active perimeter device $\frac{3}{2}c + r + 2 + j$.

The assignemt can be completed by assigning defective devices from the north-west quadrant to active perimeter devices whose number correspond to an A, north-east with B etc. Note that as these assignment can be freely done, it is possible to balance the number of defective devices assigned to active perimeter devices in the contour of the gap, and thus the assignment witnesses an optimal load.

It is easy to see that the assignment can be computed in constant time and has dilation $d = \frac{r+c}{2} + 1$.

An example of applying this scheme for a gap of 8 rows and 14 columns is depicted in fig. 5.22.

For cases 2 and 3 we give a sketch of the assignment that deals only with the most critical positions. The schemes are easy to complete but tedious to describe in details.

Case 2:
Compute $d = \frac{c+r+1}{2} + 1$.

Suppose w.l.o.g. that r is even and c is odd.

For $1 \leq i \leq r$, assign the defective device (i, c) to active perimeter device $c + 1 + i$.

Assign:

- the defective device $\left(\frac{r}{2}, \lfloor\frac{c}{2}\rfloor\right)$ to active perimeter device $\lfloor\frac{c}{2}\rfloor$;
- the defective device $\left(\frac{r}{2}, \lfloor\frac{c}{2}\rfloor + 1\right)$ to active perimeter device $\lfloor\frac{c}{2}\rfloor + d$;
- the defective device $\left(\frac{r}{2} + 1, \lfloor\frac{c}{2}\rfloor + 1\right)$ to active perimeter device $\lfloor\frac{c}{2}\rfloor + 2d - 1$;
- the defective device $\left(\frac{r}{2} + 1, \lfloor\frac{c}{2}\rfloor\right)$ to active perimeter device $\lfloor\frac{c}{2}\rfloor + 3d - 1$.

To complete the assignment, the scheme for case 1 can be applied with minor changes.

Case 3:
Compute $d = \frac{r+c}{2} + 1$.

For $1 \leq i \leq r$, assign the defective device (i, c) to active perimeter device $c + 1 + i$.
For $j \leq 1 < c$ assign the defective device (r, j) to active perimeter device $2c + r + 3 - j$.
Assign:

- the defective device $\left(\lfloor \frac{r}{2} \rfloor, \lfloor \frac{c}{2} \rfloor\right)$ to active perimeter device $\lfloor \frac{c}{2} \rfloor$;
- the defective device $\left(\lfloor \frac{r}{2} \rfloor, \lfloor \frac{c}{2} \rfloor + 1\right)$ to active perimeter device $\lfloor \frac{c}{2} \rfloor + d$;
- the defective device $\left(\lfloor \frac{r}{2} \rfloor + 1, \lfloor \frac{c}{2} \rfloor + 1\right)$ to active perimeter device $\lfloor \frac{c}{2} \rfloor + 2d$;
- the defective device $\left(\lfloor \frac{r}{2} \rfloor + 1, \lfloor \frac{c}{2} \rfloor\right)$ to active perimeter device $\lfloor \frac{c}{2} \rfloor + 3d$.

Also in this case, the assignment can be completed by applying the scheme for case 1 with minor changes.

As the dilation obtained by the assignments given by these schemes equals the lower bound given in Theorem 5.3.3, the thesis follows. □

5.3.7 Conclusion

In this section we studied the problem of bridging gaps in wireless ad-hoc lattice computers by assigning active devices on the perimeter of the gap as proxies to the defective devices in the gap. We established lower bounds on the communication dilation witnessed by such proxy assignments for single-row gaps and general row-column convex gaps. We presented dilation-optimal, constant time algorithms for computing proxy assignments for single-row gaps and gaps that are rectangular in shape. Moreover, we also showed that our proxy assignments are load-optimal.

Establishing bounds and studying the optimality of proxy assignments with respect to congestion (defined in Section 5.3.4) is an interesting open problem.

Bibliography

[1] Abiteboul, S., Kaplan, H., and Milo, T. Compact labeling schemes for ancestor queries. In *SODA* (2001), pp. 547–556.

[2] Agnarsson, G., and Halldórsson, M. M. On colorings of squares of outerplanar graphs. In *SODA* (2004), pp. 244–253.

[3] Aigner, M., and Ziegler, G. M. *Proofs from the book*. Springer-Verlag, 2004.

[4] Alon, N., Bar-Noy, A., Linial, N., and Peleg, D. A lower bound for radio broadcast. *Journal of Computer and System Sciences 43*, 2 (1991), 290–298.

[5] Ambühl, C. An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In *ICALP* (2005), pp. 1139–1150.

[6] Anastasi, G., Conti, M., Falchi, A., Gregori, E., and Passarella, A. Performance measurements of motes sensor networks. In *MSWiM* (2004), pp. 174–181.

[7] Arnborg, S., Corneil, D. G., and Proskurowski, A. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8 (1987), 277–284.

[8] Awerbuch, B., Goldreich, O., Peleg, D., and Vainish, R. A trade-off between information and communication in broadcast protocols. *Journal of the ACM 37*, 2 (1990), 238–256.

[9] Babaoglu, O., Jelasity, M., and Montresor, A. Grassroots approach to self-management in large-scale distributed systems. *Unconventional Programming Paradigms. Lecture Notes in Computer Science, Springer Verlag 3566* (2005).

[10] Bar-Yehuda, R., Goldreich, O., and Itai, A. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences 45*, 1 (1992), 104–126.

[11] Bar-Yehuda, R., Israeli, A., and Itai, A. Multiple communication in multihop radio networks. *SIAM Journal on Computing 22*, 4 (1993), 875–887.

[12] Bartolini, N., Calamoneri, T., Fusco, E. G., Massini, A., and Silvestri, S. Autonomous deployment of self-organizing mobile sensors for a complete coverage. In *IWSOS* (2008). To Appear.

[13] Bartolini, N., Calamoneri, T., Fusco, E. G., Massini, A., and Silvestri, S. Snap and spread: A self-deployment algorithm for mobile sensor networks. In *DCOSS* (2008), pp. 451–456.

[14] Bartolini, N., Calamoneri, T., Fusco, E. G., Massini, A., and Silvestri, S. Push and pull: autonomous deployment of mobile sensors for a complete coverage. *Wireless Networks* (2009). To appear.

[15] Bender, M., Fernández, A., Ron, D., Sahai, A., and Vadhan, S. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation 176* (2002), 1–21.

[16] Bodlaender, H. L. On linear time minor tests and depth first search. In *WADS* (1989), pp. 577–590.

[17] Bodlaender, H. L. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing 25*, 6 (1996), 1305–1317.

[18] Bodlaender, H. L., Kloks, T., Tan, R. B., and van Leeuwen, J. Approximations for lambda-colorings of graphs. *The Computer Journal 47*, 2 (2004), 193–204.

[19] Bonsma, P. S., Brüggemann, T., and Woeginger, G. J. A faster FPT algorithm for finding spanning trees with many leaves. In *MFCS* (2003), pp. 259–268.

[20] Brass, P. Bounds on coverage and target detection capabilities for models of networks of mobile sensors. *ACM Transactions on Sensor Networks 3* (2007).

[21] Bruce, R. J., and Hoffmann, M. $l(p, q)$-labeling of outerplanar graphs. Tech. Rep. 2003/9, Department of Mathematics and Computer Science, University of Leicester, England, 2003.

[22] Bruschi, D., and Del Pinto, M. Lower bounds for the broadcast problem in mobile radio networks. *Distributed Computing 10*, 3 (1997), 129–135.

[23] Calamoneri, T. The $l(h, k)$-labelling problem: A survey and annotated bibliography. *The Computer Journal. 49*, 5 (2006), 585–608.

[24] Calamoneri, T., Clementi, A. E. F., Fusco, E. G., and Silvestri, R. Maximizing the number of broadcast operations in static random geometric ad-hoc networks. In *OPODIS* (2007), pp. 247–259.

[25] Calamoneri, T., Fusco, E. G., and Pelc, A. Impact of information on asynchronous broadcasting. In *OPODIS* (2008). To appear.

[26] Calamoneri, T., Fusco, E. G., Shende, A. M., and Shende, S. M. Proxy assignments for filling gaps in wireless ad-hoc lattice computers. In *SIROCCO* (2007), pp. 208–221.

[27] Calamoneri, T., Fusco, E. G., Tan, R. B., and Vocca, P. L(h, 1, 1)-labeling of outerplanar graphs. In *SIROCCO* (2006), pp. 268–279.

[28] Calamoneri, T., Fusco, E. G., Tan, R. B., and Vocca, P. L(h,1,1)-labeling of outerplanar graphs. *Mathematical Methods of Operations Research*, 69 (2009). To appear.

[29] Calamoneri, T., and Petreschi, R. $l(h, 1)$-labeling subclasses of planar graphs. *Journal of Parallel and Distributed Computing 64*, 3 (2004), 414–426.

[30] Calinescu, G., Kapoor, S., Olshevsky, A., and Zelikovsky, A. Network lifetime and power assignment in ad hoc wireless networks. In *ESA* (2003), pp. 114–126.

[31] Caminiti, S., and Fusco, E. G. On the number of labeled k-arch graphs. *Journal of Integer Sequences 10*, 7 (2007).

[32] Caminiti, S., Fusco, E. G., and Petreschi, R. A bijective code for k-trees with linear time encoding and decoding. In *ESCAPE* (2007), pp. 408–420.

[33] Caminiti, S., Fusco, E. G., and Petreschi, R. Bijective linear time coding and decoding for k-trees. *Theory of Computing Systems* (2008). To appear.

[34] Caragiannis, I., Flammini, M., and Moscardelli, L. An exponential improvement on the MST heuristic for minimum energy broadcasting in ad hoc wireless networks. In *ICALP* (2007), pp. 447–458.

[35] Cardei, M., and Du, D.-Z. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks 11*, 3 (2005), 333–340.

[36] Cardei, M., Wu, J., and Lu, M. Improving network lifetime using sensors with adjustable sensing ranges. *International Journal of Sensor Networks 1*, 1/2 (2006), 41–49.

[37] Caro, Y., West, D. B., and Yuster, R. Connected domination and spanning trees with many leaves. *SIAM Journal on Discrete Mathematics 13*, 2 (2000), 202–211.

[38] Case, J., Rajan, D. S., and Shende, A. M. Lattice computers for approximating euclidean space. *Journal of the ACM 48*, 1 (2001), 110–144.

[39] Cayley, A. A theorem on trees. *Quarterly Journal of Mathematics 23* (1889), 376–378.

[40] Chen, J., Li, S., and Sun, Y. Novel deployment schemes for mobile sensor networks. *Sensors 7* (2007).

[41] Chlamtac, I., and Kutten, S. On broadcasting in radio networks - problem analysis and protocol design. *IEEE Transactions on Communications 33* (1985), 1240–1246.

[42] Chlamtac, I., and Weinstein, O. The wave expansion approach to broadcasting in multihop radio networks. *IEEE Transactions on Communications 39* (1991), 426–433.

[43] Chlebus, B. S., Gąsieniec, L., Gibbons, A., Pelc, A., and Rytter, W. Deterministic broadcasting in ad hoc radio networks. *Distributed Computing 15*, 1 (2002), 27–38.

[44] Chlebus, B. S., Gąsieniec, L., Östlin, A., and Robson, J. M. Deterministic radio broadcasting. In *ICALP* (2000), pp. 717–728.

[45] Chlebus, B. S., and Kowalski, D. R. A better wake-up in radio networks. In *PODC* (2004), pp. 266–274.

[46] Chlebus, B. S., and Rokicki, M. A. Centralized asynchronous broadcast in radio networks. *Theoretical Computer Science 383*, 1 (2007), 5–22.

[47] Chrobak, M., and Dürr, C. Reconstructing hv-convex polyominoes from orthogonal projections. *Information Processing Letters 69*, 6 (1999), 283–289.

[48] Chrobak, M., Gąsieniec, L., and Kowalski, D. R. The wake-up problem in multi-hop radio networks. In *SODA* (2004), pp. 992–1000.

[49] Chrobak, M., Gąsieniec, L., and Rytter, W. Fast broadcasting and gossiping in radio networks. In *FOCS* (2000), pp. 575–581.

[50] Chrobak, M., Gąsieniec, L., and Rytter, W. Fast broadcasting and gossiping in radio networks. *Journal of Algorithms 43*, 2 (2002), 177–189.

[51] Chu, Y. J., and Liu, T. H. On the shortest arborescence of a directed graph. *Science Sinica*, 14 (1965), 1396–1400.

[52] Clementi, A. E. F., Crescenzi, P., Penna, P., Rossi, G., and Vocca, P. On the complexity of computing minimum energy consumption broadcast subgraphs. In *STACS* (2001), pp. 121–131.

[53] Clementi, A. E. F., Huiban, G., Rossi, G., Verhoeven, Y. C., and Penna, P. On the approximation ratio of the MST-based heuristic for the energy-efficient broadcast problem in static ad-hoc radio networks. In *IPDPS* (2003), p. 222.

[54] Clementi, A. E. F., Monti, A., and Silvestri, R. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *SODA* (2001), pp. 709–718.

[55] Clementi, A. E. F., Monti, A., and Silvestri, R. Distributed broadcast in radio networks of unknown topology. *Theoretical Computer Science 302*, 1-3 (2003), 337–364.

[56] Czumaj, A., and Rytter, W. Broadcasting algorithms in radio networks with unknown topology. *Journal of Algorithms 60*, 2 (2006), 115–143.

[57] De Marco, G. Distributed broadcast in unknown radio networks. In *SODA* (2008), pp. 208–217.

[58] Dessmark, A., and Pelc, A. Optimal graph exploration without good maps. *Theoretical Computer Science 326* (2004), 343–362.

[59] Dessmark, A., and Pelc, A. Broadcasting in geometric radio networks. *Journal of Discrete Algorithms 5*, 1 (2007), 187–201.

[60] Dijkstra, E. W. Self-stabilizing systems in spite of distributed control. *Communications of the ACM 17*, 11 (1974), 643–644.

[61] Dijkstra, E. W. A belated proof of self-stabilization. *Distributed Computing 1*, 1 (1986), 5–6.

[62] Diks, K., Kranakis, E., Krizanc, D., and Pelc, A. The impact of knowledge on broadcasting time in linear radio networks. *Theoretical Computer Science 287* (2002), 449–471.

[63] Ding, G., Johnson, T., and Seymour, P. Spanning trees with many leaves. *Journal of Graph Theory 37* (2001), 189–197.

[64] Edmonds, J. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71 (1967), 233–240.

[65] Elkin, M., and Kortsarz, G. A sublogarithmic approximation algorithm for the undirected telephone broadcast problem: a path out of a jungle. In *SODA* (2003), pp. 76–85.

[66] Elkin, M., and Kortsarz, G. Improved schedule for radio broadcast. In *SODA* (2005), pp. 222–231.

[67] Elsässer, R., Gąsieniec, L., and Sauerwald, T. On radio broadcasting in random geometric graphs. In *DISC* (2008), pp. 212–226.

[68] Emek, Y., Gąsieniec, L., Kantor, E., Pelc, A., Peleg, D., and Su, C. Broadcasting in udg radio networks with unknown topology. In *PODC* (2007), pp. 195–204.

[69] Emek, Y., Kantor, E., and Peleg, D. On the effect of the deployment setting on broadcasting in euclidean radio networks. In *PODC* (2008), pp. 223–232.

[70] Fich, F., and Ruppert, E. Hundreds of impossibility results for distributed computing. *Distributed Computing 16* (2003), 121–163.

[71] Finocchi, I., Fusco, E. G., and Petreschi, R. A note on algebraic hypercube colorings. In *ITNG* (2008), pp. 869–874.

[72] Flammini, M., Navarra, A., and Pérennes, S. The "real" approximation factor of the MST heuristic for the minimum energy broadcasting. In *WEA* (2005), pp. 22–31.

[73] Fraigniaud, P., Gavoille, C., Ilcinkas, D., and Pelc, A. Distributed computing with advice: Information sensitivity of graph coloring. In *ICALP* (2007), pp. 231–242.

[74] Fraigniaud, P., Ilcinkas, D., and Pelc, A. Oracle size: a new measure of difficulty for communication tasks. In *PODC* (2006), pp. 179–187.

[75] Fraigniaud, P., Ilcinkas, D., and Pelc, A. Tree exploration with an oracle. In *MFCS* (2006), pp. 24–37.

[76] Fraigniaud, P., Korman, A., and Lebhar, E. Local MST computation with short advice. In *SPAA* (2007), pp. 154–160.

[77] Frisch, U., Hasslacher, B., and Pomeau, Y. Lattice-gas automata for the Navier Stokes equation. *Physical Review Letters 14*, 56 (April 1986), 1505–1508.

[78] Fujie, T. The maximum-leaf spanning tree problem: Formulations and facets. *Networks 43*, 4 (2004), 212–223.

[79] Fusco, E. G., and Monti, A. Spanning trees with many leaves in regular bipartite graphs. In *ISAAC* (2007), pp. 904–914.

[80] Fusco, E. G., and Pelc, A. Acknowledged broadcasting in ad hoc radio networks. *Information Processing Letters* (2008). To appear.

[81] Fusco, E. G., and Pelc, A. Broadcasting in UDG radio networks with missing and inaccurate information. In *DISC* (2008), pp. 257–273.

[82] Fusco, E. G., and Pelc, A. Trade-offs between the size of advice and broadcasting time in trees. In *SPAA* (2008), pp. 77–84.

[83] Fusco, E. G., and Pelc, A. Trade-offs between the size of advice and broadcasting time in trees. *Algorithmica* (2009). To appear.

[84] Gaber, I., and Mansour, Y. Centralized broadcast in multihop radio networks. *Journal of Algorithms 46*, 1 (2003), 1–20.

[85] Galbiati, G., Maffioli, F., and Morzenti, A. A short note on the approximability of the maximum leaves spanning tree problem. *Information Processing Letters 52*, 1 (1994), 45–49.

[86] Garey, M. R., and Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[87] Gavoille, C., Peleg, D., Pérennes, S., and Raz, R. Distance labeling in graphs. In *SODA* (2001), pp. 210–219.

[88] Gąsieniec, L. Personal communication, 2008.

[89] Gąsieniec, L., Kowalski, D. R., Lingas, A., and Wahlen, M. Efficient broadcasting in known geometric radio networks with non-uniform ranges. In *DISC* (2008), pp. 274–288.

[90] Gąsieniec, L., Pagourtzis, A., Potapov, I., and Radzik, T. Deterministic communication in radio networks with large labels. *Algorithmica 47*, 1 (2007), 97–117.

[91] Gąsieniec, L., Pelc, A., and Peleg, D. The wakeup problem in synchronous broadcast systems. *SIAM Journal on Discrete Mathematics 14*, 2 (2001), 207–222.

[92] Gąsieniec, L., Peleg, D., and Xin, Q. Faster communication in known topology radio networks. In *PODC* (2005), pp. 129–137.

[93] Goddard, W., Hedetniemi, S. T., Jacobs, D. P., and Srimani, P. K. Fault tolerant algorithms for orderings and colorings. In *IPDPS* (2004).

[94] Greenspan, D. Deterministic computer physics. *International Journal of Theoretical Physics 21*, 6/7 (1982), 505–523.

[95] Griggs, J. R., Kleitman, D. J., and Shastri, A. Spanning trees with many leaves in cubic graphs. *Journal of Graph Theory 13*, 6 (1989), 669–695.

[96] Griggs, J. R., and Wu, M. Spanning trees in graphs of minimum degree 4 or 5. *Discrete Mathematics 104*, 2 (1992), 167–183.

[97] Griggs, J. R., and Yeh, R. K. Labelling graphs with a condition at distance 2. *SIAM Journal on Discrete Mathematics 5*, 4 (1992), 586–595.

[98] Gupta, P., and Kumar, P. *Critical power for asymptotic connectivity in wireless networks*, vol. Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of W.H. Fleming. Birkhauser, Boston, 1999, pp. 547–566.

[99] Gupta, V., and Mathur, G. Wireless ad-hoc lattice computers (wadl). In *18th Annual Consortium for Computing Sciences in Colleges: Southeastern Conference* (2004).

[100] Gupta, V., Mathur, G., and Shende, A. M. Lattice formation in a wadl (wireless ad-hoc lattice computer). In *A_SWAN* (2004).

[101] Halpern, J. Y., and Moses, Y. Knowledge and common knowledge in a distributed environment. *Journal of the ACM 37*, 3 (1990), 549–587.

[102] Heo, N., and Varshney, P. Energy-efficient deployment of intelligent mobile sensor networks. *IEEE Transactions on Systems, Man and Cybernetics 35* (2005).

[103] Hillis, W. D. The connection machine: A computer architecture based on cellular automata. *Physica D 10* (1984), 213–228.

[104] Howard, A., Mataric, M. J., and Sukhatme, G. S. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *DARS* (2002).

[105] Hewlett packard: Adaptive enterprise design principles. *http://h71028.www7.hp.com/enterprise/cache/80425-0-0-0-121.html*.

[106] Huang, C.-F., and Tseng, Y.-C. The coverage problem in a wireless sensor network. *Mobile Networks and Applications 10*, 4 (2005), 519–528.

[107] Ibm: the vision of autonomic computing. *http://www.research.ibm.com/autonomic/manifesto*.

[108] Ilcinkas, D., Kowalski, D. R., and Pelc, A. Fast radio broadcasting with advice. In *SIROCCO* (2008), pp. 291–305.

[109] Jonas, K. *Graph Coloring Analogues With a Condition at Distance Two: $L(2,1)$-Labelings and List λ-Labelings*. PhD thesis, University of South Carolina, Columbia, 1993.

[110] Jurdzinski, T., and Stachowiak, G. Probabilistic algorithms for the wakeup problem in single-hop radio networks. In *ISAAC* (2002), pp. 535–549.

[111] Kang, I., and Poovendran, R. Maximizing network lifetime of broadcasting over wireless stationary ad hoc networks. *Mobile Networks and Applications 10*, 6 (2005), 879–896.

[112] Katz, M., Katz, N. A., Korman, A., and Peleg, D. Labeling schemes for flow and connectivity. In *SODA* (2002), pp. 927–936.

[113] Kerr, W., Spears, D., Spears, W., and Thayer, D. Two formal fluid models for multi-agent sweeping and obstacle avoidance. In *FAABS* (2004), pp. 111–130.

[114] Kirousis, L. M., Kranakis, E., Krizanc, D., and Pelc, A. Power consumption in packet radio networks. *Theoretical Computer Science 243*, 1-2 (2000), 289–305.

[115] Kleitman, D. J., and West, D. B. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics 4*, 1 (1991), 99–106.

[116] Korman, A., Kutten, S., and Peleg, D. Proof labeling schemes. In *PODC* (2005), pp. 9–18.

[117] Kortsarz, G., and Peleg, D. Approximation algorithms for minimum-time broadcast. *SIAM Journal on Discrete Mathematics 8*, 3 (1995), 401–427.

[118] Kowalski, D. R., and Pelc, A. Time of deterministic broadcasting in radio networks with local knowledge. *SIAM Journal on Computing 33*, 4 (2004), 870–891.

[119] Kowalski, D. R., and Pelc, A. Broadcasting in undirected ad hoc radio networks. *Distributed Computing 18*, 1 (2005), 43–57.

[120] Kowalski, D. R., and Pelc, A. Time complexity of radio broadcasting: adaptiveness vs. obliviousness and randomization vs. determinism. *Theoretical Computer Science 333*, 3 (2005), 355–371.

[121] Kowalski, D. R., and Pelc, A. Optimal deterministic broadcasting in known topology radio networks. *Distributed Computing 19*, 3 (2007), 185–195.

[122] Kranakis, E., Krizanc, D., and Pelc, A. Fault-tolerant broadcasting in radio networks. *Journal of Algorithms 39*, 1 (2001), 47–67.

[123] Kranakis, E., Paquette, M., and Pelc, A. Communication in networks with random dependent faults. In *MFCS* (2007), pp. 418–429.

[124] Kuba, A. The reconstruction of two-directional connected binary patterns from their two orthogonal projections. *Computer Vision, Graphics, Image Processing* (1984), 249–265.

[125] Kushilevitz, E., and Mansour, Y. An Omega($D \log N/D$) lower bound for broadcast in radio networks. *SIAM Journal on Computing 27*, 3 (1998), 702–712.

[126] Lemke, P. The maximum leaf spanning tree problem for cubic graphs is *np*-complete. Tech. Rep. IMA Preprint Series # 428, University of Minnesota, July 1988.

[127] Li, P. C., and Toulouse, M. Variations of the maximum leaf spanning tree problem for bipartite graphs. *Information Processing Letters 97*, 4 (2006), 129–132.

[128] Lloyd, E. L., and Ramanathan, S. On the complexity of distance-2 coloring. In *ICCI* (1992), pp. 71–74.

[129] Lorys, K., and Zwozniak, G. Approximation algorithm for the maximum leaf spanning tree problem for cubic graphs. In *ESA* (2002), pp. 686–697.

[130] Lu, H.-I., and Ravi, R. The power of local optimizations: Approximation algorithms for maximum-leaf spanning tree (draft)*. Tech. Rep. CS-96-05, 1996.

[131] Lu, H.-I., and Ravi, R. Approximating maximum leaf spanning trees in almost linear time. *Journal of Algorithms 29*, 1 (1998), 132–141.

[132] Lynch, N. A hundred impossibility proofs for distributed computing. In *PODC* (1989), pp. 1–28.

[133] Lynch, N. A. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[134] Ma, M., and Yang, Y. Adaptive triangular deployment algorithm for unattended mobile sensor networks. *IEEE Transactions on Computers 56* (2007).

[135] Margolus, N. CAM-8: a computer architecture based on cellular automata. In *Fields Institute Communications - Pattern Formation and Lattice-Gas Automata* (1993), A. Lawniczak and R. Kapral, Eds.

[136] McCormick, S. T. Optimal approximation of sparse hessians and its equivalence to a graph coloring problem. *Mathematical Programming 26* (1983), 153–171.

[137] Microsoft: The drive to self-managing dynamic systems. *http://www.microsoft.com/windowsserversystem/dsi/default.mspx*.

[138] Moore, P., and Shende, A. M. Gaps in wireless ad-hoc lattice computers. In *International Symposium on Wireless Communication Systems* (2005).

[139] Nisse, N., and Soguet, D. Graph searching with advice. In *SIROCCO* (2007), pp. 51–65.

[140] Opnet technologies inc. *http://www.opnet.com*.

[141] Pac, M. R., Erkmen, A. M., and Erkmen, I. Scalable self-deployment of mobile sensor networks; a fluid dynamics approach. *IROS* (2006).

[142] Pahlavan, K., and Levesque, A. H. *Wireless information networks.* Wiley-Interscience, New York, NY, USA, 1995.

[143] Pattem, S., Poduri, S., and Krishnamachari, B. Energy-quality tradeoffs for target tracking in wireless sensor networks. In *IPSN* (2003), pp. 32–46.

[144] Pelc, A. Fault-tolerant broadcasting and gossiping in communication networks. *Networks 28*, 3 (1996), 143–156.

[145] Pelc, A. Activating anonymous ad hoc radio networks. *Distributed Computing 19* (2007), 361–371.

[146] Penrose, M. D. *Random Geometric Grpahs.* Oxford University Press, 2003.

[147] Poduri, S., and Sukhatme, G. S. Constrained coverage for mobile sensor networks. *ICRA* (2004), 165–171.

[148] Proskurowski, A. Minimum broadcast trees. *IEEE Transactions on Computers 30*, 5 (1981), 363–366.

[149] Rahman, M. S., and Kaykobad, M. Complexities of some interesting problems on spanning trees. *Information Processing Letters 94*, 2 (2005), 93–97.

[150] Ravishankar, K., and Singh, S. Broadcasting on $[0, l]$. *Discrete Applied Mathematics 53* (1994), 299–319.

[151] Robertson, N., and Seymour, P. D. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms 7*, 3 (1986), 309–322.

[152] Robertson, N., and Seymour, P. D. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B 41*, 1 (1986), 92–114.

[153] Robertson, N., and Seymour, P. D. Graph minors. iv. tree-width and well-quasi-ordering. *Journal of Combinatorial Theory, Series B 48*, 2 (1990), 227–254.

[154] Santi, P., and Blough, D. M. The critical transmitting range for connectivity in sparse wireless ad hoc networks. *IEEE Transactions on Mobile Computing 2*, 1 (2003), 25–39.

[155] Sen, A., and Huson, M. L. A new model for scheduling packet radio networks. In *INFOCOM* (1996), pp. 1116–1124.

[156] Shende, A. M. *Digital analog simulation of uniform motion in representations of physical n-space by lattice-work mimd computer architectures.* PhD thesis, SUNY, Buffalo, 1991.

[157] Slater, P. J., Cockayne, E. J., and Hedetniemi, S. T. Information dissemination in trees. *SIAM Journal on Computing 10*, 4 (1981), 692–701.

[158] Solis-Oba, R. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In *ESA* (1998), pp. 441–452.

[159] Storer, J. A. Constructing full spanning trees for cubic graphs. *Information Processing Letters 13* (1981), 8–11.

[160] Thorup, M., and Zwick, U. Approximate distance oracles. *Journal of the ACM 52*, 1 (2005), 1–24.

[161] Uchida, J., Chen, W., and Wada, K. Acknowledged broadcasting and gossiping in ad hoc radio networks. *Theoretical Computer Science 377*, 1-3 (2007), 43–54.

[162] Wan, P.-J., Calinescu, G., Li, X.-Y., and Frieder, O. Minimum-energy broadcast routing in static ad hoc wireless networks. In *INFOCOM* (2001), pp. 1162–1171.

[163] Wang, G., Cao, G., and La Porta, T. Proxy-based sensor deployment for mobile sensor networks. *MASS* (2004), 493–502.

[164] Wang, G., Cao, G., and La Porta, T. Movement-assisted sensor deployment. *IEEE Transaction on Mobile Computing 6* (2006).

[165] Wieselthier, J. E., Nguyen, G. D., and Ephremides, A. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *INFOCOM* (2000), pp. 585–594.

[166] Yeh, R. K. A survey on labeling graphs with a condition at distance two. *Discrete Mathematics 306*, 12 (2006), 1217–1231.

[167] Yepez, J. Lattice-gas dynamics, volume i viscous fluids. Tech. Rep. 1200, Phillips Laboratories, November 1995.

[168] Zhang, H., and Hou, J. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc & Sensor Wireless Networks 1*, 1-2 (2005).

[169] Zhou, X., Kanari, Y., and Nishizeki, T. Generalized vertex-coloring of partial k-trees. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E83-A (2000), 671–678.

[170] Zou, Y., and Chakrabarty, K. Sensor deployment and target localization based on virtual forces. *INFOCOM* (2003).