Problems on Geometric Graphs with Applications to Wireless Networks

by

Yurai Núñez-Rodríguez

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
November 2009

Abstract

It is hard to imagine the modern world without wireless communication. Wireless networks are, however, challenging inasmuch as they are useful. Because of their complexity, wireless networks have introduced a myriad of new problems into the field of algorithms. To tackle the new computational challenges, specific models have been devised to suit wireless networks. Most remarkably, wireless networks can be modelled as geometric graphs. This allows us to address problems in wireless networks using tools from the fields of graph theory, graph algorithms, and computational geometry. Our results have applications to routing, coverage detection, data collection, and fault recovery, among other issues in this area.

To be more specific, we investigate three major problems: a geometric approach to fault recovery; the distributed computation of Voronoi diagrams; and finding Hamiltonian cycles in hexagonal networks. Our geometric approach to fault recovery has been experimentally proved superior to an existing combinatorial approach. The distributed algorithm we propose for computing Voronoi diagrams of networks is the first non-trivial algorithm that has been proved to perform this task correctly; its efficiency has been demonstrated through simulations. Regarding the Hamiltonian cycle problem of hexagonal networks, we have advanced this topic by providing conditions for the existence of such a cycle, in addition to new insight on the complexity for the "solid" hexagonal grid case. Although we present satisfying solutions to several of the addressed problems, plenty is left to be done. In this regard, we identify a set of open problems that will be subject of research for years to come.

Acknowledgments

I could not have had better supervisors than Henk and David. Their clever way of thinking and approach to research has been a great example and guide. I deeply appreciate their availability for answering all my questions, even while being away. In particular, I would like to thank David, for trusting me from our first meetings in Havana, for accepting me as student, for his encouragement throughout these years, and for his great music.

I am very thankful to my lab mates, coauthors, and other researchers I have worked with. I have learnt a bit from every one of them. I would like to especially thank Prof. Selim Akl, Prof. James Stewart, Prof. Tetsuo Asano, Prof. Jit Bose and other researchers at Carleton University, as well as my lab mates Kamrul and Henry.

I would like to thank the staff and friends from the School of Computing with whom I have shared great moments during the coffee breaks, grad club gatherings, softball games, recreational badminton, etc.

To my parents, whom I dedicate this thesis, thanks for teaching me your moral values, for your trust and understanding, and for your prayers. And finally to Min-Ching, who is also writing her thesis as I write mine, thank you for your support and love.

Statement of Originality

I hereby certify that all the results included in the present work are original and that all ideas attributed to others have been properly acknowledged.

Professors David Rappaport and Henk Meijer, my supervisors, contributed throughout all the creative process with research directions, ideas for the proofs, and preparation of the manuscripts. Prof. Jit Bose, contributed with ideas to prove the communication lower bound for the distributed computation of Voronoi diagrams and the worst case optimal algorithm, namely the Leader-based algorithm. Prof. Chris Umans clarified some ideas about his paper on Hamiltonian cycles of solid square grids that were used for constructing Figure 4.9. Waleed Alsalih, Kamrul Islam, and Henry Xiao participated in the brainstorming that originated the work on Voronoi diagrams. With them I co-authored previous papers on this topic and through their research I learnt about several of the publications cited herein. Henry Xiao also programmed an existing distributed algorithm for computing Voronoi diagrams, which allowed for experimental comparisons. Rodoslaw Cymer pointed out previous results on factors of graphs that I was not aware of. These results have been cited in Section 4.4.2.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wireless communication networks, along with the Internet, have revolutionized and continue to change modern life as a whole. This field has received extraordinary attention from the media and the research community. As a result, it is one of the fastest growing areas within communications, as expressed by Goldsmith [31] and Walrand and Varaiya [85] in their books. Typical examples of wireless networks are remote sensing networks and mobile networks. These have applications in many fields including personal communication, environmental monitoring, medical (patient) monitoring, and military surveillance. Wireless networks typically consist of autonomous nodes with the capability of (wirelessly) communicating, performing computation, and, in some cases, moving and sensing. The possibility of communicating in a wireless manner provides additional degrees of freedom to wireless networks, allowing them to be deployed more easily, have lower maintenance requirements, and offer better adaptability and scalability than wired networks. Ad hoc networks and wireless sensor networks have stretched these capabilities to the limit, bringing about a diverse set of challenges. Multiple research areas come together to address such

challenges, most notably, electronics, networking, and algorithms.

Because of the geometric nature of wireless networks, many of the problems that originate from them can be solved using techniques from the field of computational geometry. The present work proposes new geometric tools for tackling problems such as fault recovery, data collection, coverage, and routing in wireless networks. For this, we study problems on algorithms and combinatorics with an underlying geometric component. The problems studied herein are interesting from a theoretical point of view, in addition to their practical implications.

It is also important to highlight the role of distributed computation in wireless networks. Distributed computation allows a divide-and-conquer approach to networks problems, in many cases minimizing the use of resources such as computer memory, time, and energy. Some of the problems we address in this thesis will be solved in a distributed manner. The extent to which each problem can be distributed will be discussed in the corresponding chapter.

1.1 The Problems of Study

We investigate problems related to fault recovery, routing, data collection, and coverage in wireless networks. Some of these problems are addressed directly, whereas others have a less direct relationship with our work. We consider networks embedded in a two-dimensional Euclidean space, that is, the real plane.

We study a fault recovery technique that allows nodes to correct data that may have been corrupted due to some kind of (temporary) fault. To this end we propose a geometric recolouring technique that has been used before for preprocessing geographic data. Part of our research in this direction is devoted to obtaining tight

combinatorial bounds on the geometric recolouring process itself.

Voronoi diagrams (VDs) are a meaningful way to partition the space into regions such that each region contains the node that defines it. VDs are formally defined in Chapter 3 (see Figure 3.1). They allow efficient algorithms for determining coverage. Also, their dual graphs, the Delaunay triangulations, are very efficient for routing purposes. We study the problem of efficiently computing VDs of connected wireless networks. According to the existing literature, previous approaches either give approximate solutions, work only for specific types of networks, or do not include proofs of correctness or complexity.

The Hamiltonian circuit (HC) problem is largely known to be NP-Complete for general graphs and even special types of graphs, such as planar graphs and grid graphs. We study the HC problem in solid hexagonal grid graphs. In previous work [37], we proved that the HC problem is NP-Complete for general hexagonal grid graphs. Solid hexagonal grid graphs have applications in wireless networks as they define a natural configuration of nodes that "covers well" the network region. We also study a problem closely related to the HC, namely, k-factors in graphs.

1.2 Contributions

The main contributions of this thesis are listed below along with each one of the addressed problems. Notice that for the non-specialized reader to understand some of these concepts, it may be necessary to read the corresponding chapters.

- Fault recovery in wireless networks: the geometric recolouring approach. We present tight bounds for the length of geometric recolouring sequences in various

types of geometric graphs, including an upper bound pertaining to triangulations that had been posted as an open problem (see [72]). A novel strategy for wireless networks to recover from software faults was devised based on geometric recolouring. The effectiveness of our strategy was demonstrated through experiments.

- Distributed network computation of Voronoi diagrams. We proved tight lower bounds on the communication complexity of this problem. A worst-case optimal algorithm meeting the lower bounds, the Leader-based algorithm, is proposed. In addition, we present a more practical algorithm, the CCCG algorithm, that has been experimentally shown to perform optimally on average for realistic networks. We provide formal proofs that our algorithm correctly computes the Voronoi diagram for any connected network.

- Hamiltonian cycles in solid hexagonal grids. This problem is known to have polynomial time solution for the square grid case, as proposed by Umans and Lenhart [83]. We examine their approach and conclude that there are many more cases to consider for solid hexagonal grids. Necessary conditions were proved to aid at solving this problem. Another problem related to the Hamiltonian cycle, construction of k-factors, is also examined. We point out the problems with several existing algorithms to compute k-factors and propose a more suitable one.

1.3 Structure of the Thesis

The next section defines preliminary concepts required for the understanding of our proposed work. Chapter 2 presents a study of geometric recolouring and its applications to fault recovery in wireless networks. Chapter 3 presents the problem of computing the Voronoi diagram of a connected network in a distributed manner. Chapter 4 addresses the problem of deciding whether a solid hexagonal grid graph has a Hamiltonian circuit. Literature reviews for each one of these topics will be given as part of the corresponding chapter. Each chapter also contains a set of open problems and concluding remarks on the topic it addresses. At the end of this thesis we include a summary of results.

1.4 Preliminaries

In order to put our work in context, we first characterize different types of wireless networks and distinguish those to which our results apply best. This is the subject of Section 1.4.1. The two sections that follow describe two different models of wireless networks: one from a geometric perspective and the other from a communication network perspective. These two models, or the combination of them, will be used as a platform for our algorithms and their analysis.

1.4.1 Wireless Networks

The term *wireless network* in general refers to any network of devices that communicate in a wireless manner. According to the type of device and communication, wireless networks can be very diverse. There are, for example, computer networks (of

local or wide scope), cellular networks, and satellite networks. These involve a set of heterogeneous devices such as routers, base stations, computers, satellite receivers, and mobile phones. Although our results apply to these and any other type of network that can be modelled as a geometric graph (see Section 1.4.2), we concentrate on the study of dynamic wireless networks of homogeneous computing devices. Such devices can communicate (in a wireless manner), can be mobile, are battery-operated, and have limited storage capacity (memory). These are characteristics of mobile ad hoc networks and wireless sensor networks. In the following we refer to the devices, or wireless network elements, as nodes.

A typical form of wireless communication among nodes is through radio signals; although other media, such as sound or light, are also used in some cases. The signal emitted by a node can reach a set of points in the space around itself, the so-called *coverage area*. The coverage area is sometimes modelled as a circle around the node defined by its *transmission radius*.

Mobile ad hoc networks (MANETs), as described by Sarkar, Basavaraju, and Puttamadappa in their book [73], are wireless networks that form dynamically and temporarily, without any existing infrastructure, in which all mobile nodes also play the role of routing. Thus, for a message to go from a source node to a destination node, it may require multiple hops along a dynamic path formed by other nodes. The topology of a MANET can change very rapidly and unexpectedly as a consequence of the nodes' mobility. In general, MANETs are very appealing because of their simplicity and easy deployment.

Wireless sensor networks (WSNs) consist of tiny battery-operated devices (sensor

nodes) that have the additional capability of sensing. There are sensors for temperature, light, acoustics, and specific chemicals, to mention but a few. In most of their aspects WSNs behave just like MANETs; although, unlike MANETs, WSNs nodes may have fixed locations. WSNs, however, are also very dynamic, considering other factors, such as energy depletions and node failures. The latter is magnified for WSNs deployed in hazardous environments (see [86] for example). Several problems related to wireless sensor networks are surveyed in Zhao and Guibas [87].

Because of their characteristics, WSNs and MANETs require algorithms that rely as little as possible on the network structure. This idea is reflected in the concept of *locality*, as will be defined in Section 1.4.3. It is also important to design algorithms that minimize the network traffic, because communication is the most expensive part of distributed computation in terms of time and energy consumption.

Among the most common functions of wireless network are: *querying*, that is, requesting some information at a certain node that may be available at the node itself, some other node in the network, or scattered throughout the network; *routing*, which consists of forwarding a message to its destination or relaying it to another node that is closer to the destination according to some criteria; and *topology control*, in which a node decides, among the nodes it can directly communicate with, which ones to talk to or consider for a certain purpose. These functions may be closely related to one another. For example, topology control may be a preprocessing step for efficient routing, and routing is obviously an important element in querying the network. The problems we address in the present work are related to topology control, routing, and querying. We also propose another functionality that allows nodes to recover from faults or corrupted information. Although *fault recovery* is not an entirely new idea,

our approach significantly differs from previous approaches (see Section 2.4).

In the specific case of wireless sensor networks, our work is relevant to other functions: these are *scheduling*, which defines which nodes must be active at a certain time for the network to accomplish a certain task, while the rest can remain inactive and save battery power that way; *sensing*, which controls parameters for the sensing activity, such as the direction of (directional) antennas; and *target tracking*, which involves a large set of other functions such as scheduling and sensing, with the purpose of keeping track of objects within the sensed region.

Other functions that are commonly performed in wireless networks, but that are not directly related to our work include: *localization*, in which the nodes try to discover their spatial location by using their relative position with respect to other nodes; *synchronization*, that allows nodes to perform an orchestrated distributed computation; and *security*, consisting of an assortment of tools such as cryptography, topology control, and intrusion detection.

We also use the terms *data collection*, as a way of gathering information for querying, and *coverage* which refers to whether there are points within the area of interest in which no communication signal is received or cannot be sensed by any node.

1.4.2 Geometric Graphs

A *geometric graph* consists of vertices that are points in the plane (or some other geometric space) and edges that are straight-line segments connecting the vertices. We define a network $G = (N, L)$ to be a geometric graph with vertex set N defined by a set of *nodes* embedded in the plane and edge set L consisting of straight-line *links* connecting pairs of nodes. We refer to a pair of nodes that share a link as neighbours.

G is assumed to be undirected, as is common for many network connectivity models, that is, any two neighbours $p, q \in N$ can directly communicate in both directions. The link between p and p is denoted $\overline{pq}$, or equivalently $\overline{qp}$.

A *plane geometric graph*, or *plane network*, is a network in which no two links cross. The bounded areas defined by the edges of a planar network are called *faces*. Geometric planar graphs have one unbounded face, the *outer face*, and zero or more bounded faces, the *inner faces*. The following are other types of geometric graphs that will be referred to later on.

Definition 1. ***(unit disk graph (UDG))*** *Let $G = (N, L)$ be a geometric graph. G is a* unit disk graph *if for every pair of nodes $p, q \in N$, $\overline{pq} \in L$ if and only if $\|p - q\|_2 \leq 1$.*

In the previous definition, nodes p and q are considered as vectors representing their Cartesian coordinates and $\|.\|_2$ is the norm 2 or Euclidean norm of a vector. UDGs are a very natural way to model a wireless network that uses radio signals for communication and where all nodes transmit with equal power. This type of graph is widely used in the networking literature (see [66, 10, 57] for examples). Figure 1.1 shows an example UDG with a circle marking the transmission range of node p. Nodes that fall inside the circle are neighbours of p. Although the communication distance in the definition of UDG is assumed to be one, this can be generalized to any other predetermined (unitary) distance. Notice that links in UDGs are undirected. We also refer to the UDG of a node set N as the induced UDG of N, or $UDG(N)$, meaning that the network is determined by the node's embedding and the distances between pairs of nodes. For other definitions and applications of UDGs the reader is referred to the article by Clark, Colbourn, and Johnson [18].

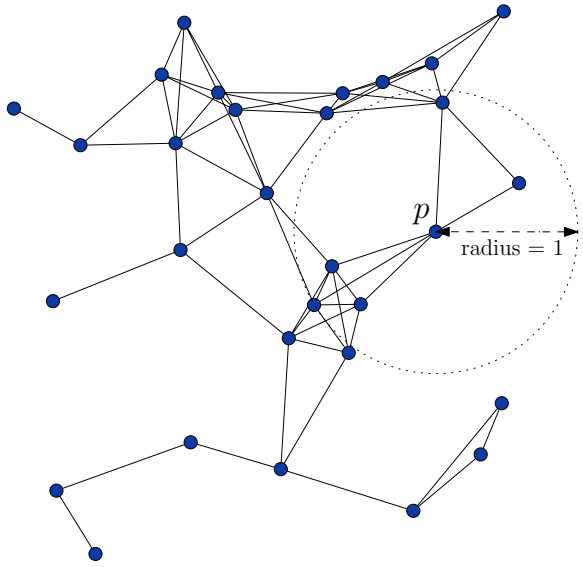

Figure 1.1: Example unit disk graph.

Definition 2. ***(t-spanner)*** *(adapted from [54]) Let N be a set of nodes embedded in the plane. A* t-spanner *of N is a network $G = (N, L)$ such that for any two nodes $p, q \in N$ there is a path in G between p and q whose length is less than or equal to $t\|p - q\|_2$.*

Definition 3. ***(stretch factor)*** *(adapted from [54]) Let $G = (N, L)$ be a geometric graph. The* stretch factor *of G is the smallest real number t such that G is a t-spanner of N.*

The concept of a spanner is useful for defining sparse networks (i.e., networks with a small number of links) that approximate more dense networks, or the complete network for that matter, in terms of distances between nodes. They have the property of providing an efficient routing infrastructure for networks: the shortest path a message needs to travel to achieve its destination on a spanner network is at most t times greater than the optimum (straight-line distance).

Other special classes of geometric graphs are regular grid graphs, as defined next.

Definition 4. ***(square grid)*** *(adapted from [4]) Let $\mathbb{Z}_{\square}$ be the infinite square (integer) lattice, i.e., the set of vertices of the tiling of the plane with unit squares. A* square grid graph*, or* square grid*, is a UDG with finite node set $N \subset \mathbb{Z}_{\square}$.*

Similarly we define triangular and hexagonal grids.

Definition 5. ***(triangular grid and hexagonal grid)*** *(adapted from [4]) Let $\mathbb{Z}_{\triangle}$ (resp. $\mathbb{Z}_{\hexagon}$) be the vertices of the tiling of the plane with unit-side equilateral triangles (resp. regular hexagons). A* triangular grid graph*, or* triangular grid*, (resp.* hexagonal grid graph*, or* hexagonal grid*) is a UDG with finite node set $N \subset \mathbb{Z}_{\triangle}$ (resp. $N \subset \mathbb{Z}_{\hexagon}$).*

We use the term *grid graph* to generically refer to a grid of either type, let it be square, triangular, or hexagonal. We refer to the type itself as *grid type*. There exists no other type of regular grid graph in the two-dimensional Euclidean space than these.

For the next definition we use the term 2-*connected*. A graph is said to be 2-connected if there is no single vertex whose removal disconnects the graph.

Definition 6. ***(solid grid)*** *Let G be a grid graph (of any class). G is a* solid grid graph *if it is 2-connected and all faces, excepting the outer face, are of minimal length, i.e., lengths 3, 4, and 6, for triangular, square, and hexagonal grids, respectively.*

See Figure 1.2 for example regular grids of all grid types. Grid graphs are used for approximating general geometric graphs. A general geometric graph may have edge lengths that are irrational or cannot be represented with total accuracy on a

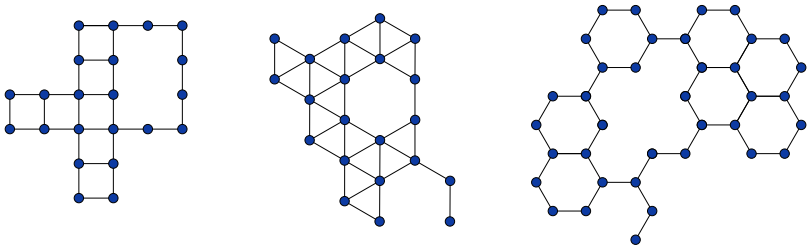

Figure 1.2: Example regular grids: (left) square, (centre) triangular, (right) hexagonal.

computer numerical system. These can be approximated by a grid graph in a similar way a straight line is represented on a raster display. By extending the definition of t-spanner, we can say that there exists a t-spanner grid graph approximation for every geometric graph G, not with respect to the Euclidean distance between nodes, but with respect to the shortest path in G. The constant t depends on the grid type. There are other applications for grid graphs; some of them will be reviewed in Chapter 4.

Definition 7. ***(triangulation)*** *A* triangulation *of a set of nodes N is a plane geometric graph $G = (N, L)$ with maximal number of links ($|L|$), i.e., no more links can be added without two of them crossing.*

In a triangulation all internal faces are triangles, therefore its name. The outer face of a triangulation of a set of nodes defines the smallest convex polygon that contains the nodes, which is called the *convex hull* of the set. Next we describe an important class of triangulations.

Definition 8. ***(Delaunay triangulation (DT))*** *The* Delaunay triangulation *of a set of nodes N is a triangulation for which the circumcircle of every inner face contains no node in its interior. This property that characterizes DTs is called the* empty circle property.

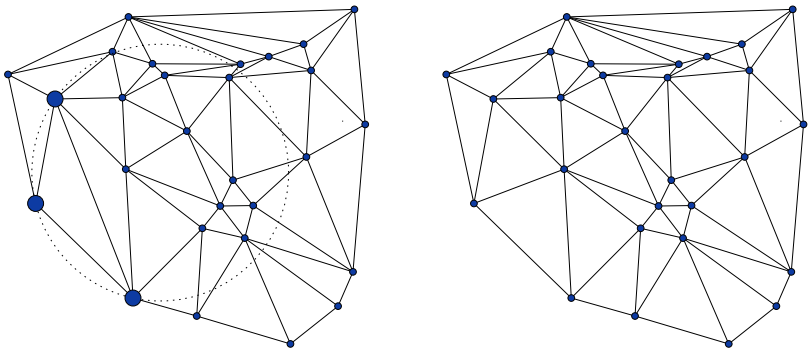

Figure 1.3: Example triangulations: (left) a non-Delaunay triangulation, (right) Delaunay triangulation.

Figure 1.3 shows an example triangulation that is not Delaunay (left) –notice that the circle defined by the nodes represented as large disks is not empty– and a Delaunay triangulation (right). DTs have nice properties such as maximizing the minimum internal angle of all (triangular) inner faces. A DT is also a spanner of its set of nodes, with stretch factor less than 2.5. A tight upper bound on the stretch factor of an arbitrary DT is unknown. (See the open problems proposed by Narasimhan and Smid [54].)

The DT of a set of nodes N is unique if all nodes in N are in a *general position*. We define general position for a two-dimensional set of nodes to mean that no three nodes are co-linear and no four nodes are co-circular.

This brief introduction to geometric graphs is not exhaustive. We present only the concepts that are necessary for the understanding of this thesis. For a more complete survey on some geometric graph topics, the reader is referred to Pach's survey [33] (Chapter 10), and his book [60] for a selection of interesting topics in this area. Moreover, as geometric graphs are present in most of computational geometry, a more exhaustive survey on this topic would be the handbook on computational

geometry edited by Goodman and O'Rourke [33].

1.4.3 The Distributed Network Model of Computation

All functions performed on a network are driven by *distributed algorithms* and their constituent *protocols*. Distributed algorithms guarantee global coordination for the completion of a given task. An algorithm may consist of one protocol executed at all nodes or several different protocols executed at specific nodes. For the design and analysis of distributed algorithms in networks the following computational model has been devised.

The *distributed network model of computation*, as defined by Peleg [65], embodies distributed computation in networks with arbitrary topology, i.e., networks that define an arbitrary geometric graph. This model reflects the problems associated with communication as the bottleneck for distributed computation. Peleg defines *locality* as the sufficiency of a node to communicate with other nodes in a bounded vicinity around itself for solving a given problem. The algorithms that exhibit this property are referred to as *localized*.

In order to analyze the complexity of communication for a distributed algorithm one must consider the limitations on the bandwidth of the network. According to the distributed network model all messages between neighbours have the same unit cost and can be of size at most $s \in O(\log n)$ bits, where n is the number of nodes in the network. Thus, whenever a message of size $L > s$ is sent between neighbours, it costs $\lceil L/s \rceil$ as it needs to be fragmented into that many messages. If the message needs to travel h hops to reach its destination, the total cost incurred is $h\lceil L/s \rceil$.

The distributed network model can be either considered as *synchronous* or *asynchronous*. In a synchronous model the computation is assumed to occur at well distinguished rounds. Each round has three phases: (1) the nodes send out messages to some of their neighbours; (2) the nodes receive messages from some of their neighbours; and (3) the nodes perform some local computation. It is assumed that the time spent on local computations, as well as the propagation delay of messages, is relatively small with respect to the waiting times, and therefore negligible. Each round must be allocated a certain amount of time for the three phases to complete, which is considered as the time unit. Some nodes may not receive or send any message during a certain round, so they sit idle waiting for a future round of messages. Thus, the computation time on the synchronous model is measured by the number of rounds it takes for the algorithm to terminate.

In contrast, in the asynchronous model nodes compute as messages arrive. This usually shortens the computation time. However, an extra degree of complexity is introduced with respect to the synchronous model, as little can be assumed regarding the order in which messages are received. In this model, time is measured in terms of real time units; however, all message interleavings must be considered in order to derive time complexity upper bounds. Therefore, although more constrained, it is easier to derive time complexity bounds on the synchronous model.

Chapter 2

Fault Recovery: The Recolouring Approach

2.1 Introduction

Faults are an inherent aspect of distributed systems of computation and communication. Some faults, such as fatal hardware crashes, are irreversible; whereas other faults, such as corrupted software, can be corrected upon detection. For example, a distributed computation network can recover from faults when duplicates of a given piece of data are stored on multiple nodes across the network. The simplest consistency rule one could impose on a piece of data in a distributed system is based on separate bits. Therefore, a node may be healthy or faulty, according to a certain bit, depending on its value. Our goal is to design wireless networks that autonomously detect and recover from faults.

In what follows we refer to faulty nodes as red and healthy nodes as blue, just to assume a convention. Assigning colours to nodes will simplify our presentation and

will put our work in the same context as previous work. Thus, the change of status of a node (from faulty to healthy, or vice verse) is called a *recolouring*.

Next we review the literature on distributed system fault recovery techniques and the origins of a technique we propose for fault recovery, namely, geometric recolouring. In Section 2.3 we present our results in geometric recolouring. These include a number of combinatorial bounds that are of theoretical interest in their own right. Section 2.4 proposes the application of geometric recolouring to fault recovery in wireless networks and compares it to more conventional techniques used for this purpose.

2.2 Literature Review

We first survey previous works on fault recovery in distributed systems. Then we introduce a motivating problem, red-blue separation, that leads to the geometric recolouring technique and its application to fault recovery.

2.2.1 Background on Fault Recovery

Several studies on fault recovery have focused on distributed systems with regular topology. For example, de La Noval et al. [21] and Floccini et al. [25] consider fault recovery on toroidal meshes and Luccio, Pagli, and Sanossian [47] study this problem on butterfly networks. Their work focuses on finding *monopolies*, that is, configurations of faulty nodes that can cause the entire network to fail. As will be seen later on, we are more concerned with the length of recolouring sequences. The common denominator of both efforts remains to study systems that can autonomously recover from faults through localized algorithms.

It is often assumed that faults occur as a consequence of manufacturing defects or other random, spontaneous causes. This is the case considered by Krishnamachari and Iyengar [44], and Luo, Dong, and Huang [48]. To our knowledge, no previous work has considered the occurrence of failures in correlation with the geometric location of the nodes. This is the case when all the nodes within a geographic area fail at a given time and for the same reason. For example, a set of nodes may temporarily fail under the influence of an electromagnetic field caused by lightning or some kind of interference, or may run out of batteries after a long period in a shadowed area without being able to recharge their batteries with the aid of solar cells.

It is widely accepted that the data collected by the nodes of a network, in the case of sensor networks for instance, is correlated to their geographic location [78, 1]; thus, there is no reason not to believe that errors induced on the network by the influence of the environment are also correlated to their spatial distribution. Notice also that considering faulty areas is a generalization to considering isolated errors. The latter can be seen as independent faults on areas that are small enough to contain a single node.

It is assumed that a node does not know whether it is faulty or not by just reading the data stored in it. It can however, compare its colour with its neighbours' colours. All the previously cited approaches to fault recovery and other studies (see the survey by Peleg [64]) use a *majority voting* rule. The majority voting rule recolours a node if it has more neighbours of the opposite colour than neighbours of its own colour. This strategy can obviously turn healthy nodes to faulty as much as healing faulty ones. However, based on the fact that faulty nodes should be a "non-dominant minority", we expect that the cooperative effort makes progress towards healing the

faulty nodes. On the other hand, we argue that the majority rule is not the best approach if applied to geometric graphs involving areas of faulty nodes. We propose a geometric recolouring approach to this end.

2.2.2 Background on Red-Blue Separation

Given a set of nodes partitioned into red and blue subsets, a red-blue separator is a boundary that separates the red points from the blue ones. There has been considerable investigation of methods for obtaining such red-blue separating boundaries. In the following we refer to points embedded in the plane, as opposed to nodes. This is the conventional terminology for the red-blue separation literature. In subsequent sections we will re-adopt the terms *nodes* and *links*, which is more natural for wireless networks.

In his PhD thesis, Seara [74], examines various means for red-blue separation, including all feasible red-blue separations by a line, a strip, or a wedge. For the case of red-blue separation with the minimum perimeter polygon the problem is known to be NP-hard [23, 5]. A somewhat related topic is to obtain a balanced subdivision of red and blue points, that is, faces of the subdivision containing a prescribed ratio of red and blue points. Kaneko and Kano [39] give a comprehensive survey of results pertaining to red and blue points in the plane, including results on balanced subdivisions.

For some applications one is willing to reclassify points by recolouring them so as to obtain a more reasonable boundary. For example, Chan [14] shows that finding a red-blue separating line with the minimum number of reclassified points takes $O((n + k^2) \log k)$ expected time, where n is the overall number of points and k is the number

of recoloured points.

Reinbacher et al. [72] study algorithms for delineating polygonal regions which are determined by coloured points. They propose a recolouring, or reclassification, method for obtaining better delineating boundaries. As a first step of the heuristic algorithm a Delaunay triangulation of the points is used to specify a neighbour relation for the points. They then define a point, p, as *surrounded* when there is a contiguous set of oppositely coloured neighbours of p, in the triangulation, that span a radial angle greater than 180°. Points that are surrounded are recoloured iteratively in no particular order until no point remains surrounded. Using properties of the triangulation Reinbacher et al. show that these recolouring sequences are finite and are guaranteed to terminate in at most $2^n - 1$ iterations. Furthermore, Reinbacher et al. demonstrate, through experimental results, that this process does a good job at reclassifying points yielding boundaries that are more suitable for their application.

2.3 Geometric Recolouring

Consider a network $G = (N, L)$ with n nodes. Our results begin where Reinbacher et al. [72] leave off. Using some of their ideas we are able to obtain an $O(n^2)$ upper bound for the number of recolourings if G is a triangulation. We also provide bounds for recolouring sequences in other network topologies. Table 2.1 summarizes our results on different types of geometric graphs. The column "All recoloured" of the table indicates whether there is a recolouring sequence that recolours all nodes.

In the next section we formally describe the recolouring problem and present some basic results obtained by Reinbacher et al. Section 2.3.2 precisely describes the recolouring problem in triangulations, including the lower bound by Reinbacher et al.

Table 2.1: Summary of main results on geometric recolouring.

Type of graph	**Lower bound**	**Upper bound**	**All recoloured**	**Section**
Triangulations	$O(n^2)$	$O(n^2)$	No	2.3.2
Max. degree three	$O(n)$	$O(n)$	No	2.3.3
Trees	$O(n)$	$O(n)$	No	2.3.6
Plane	∞	–	Unknown	2.3.4
Non-isolated convex	$O(n^2)$	$O(n^3)$	No	2.3.4
Non-plane	∞	–	Yes	2.3.5
Non-plane containing triangulation	$O(n^2)$	$O(n^3)$	No	2.3.5
One-bend plane	∞	–	Yes	2.3.7

and the new (tight) $O(n^2)$ upper bound. In Section 2.3.3 bounds are given for other types of networks, such as plane networks, non-plane networks, and trees. The last section discusses some extensions of our results.

2.3.1 Preliminaries on Geometric Recolouring

Geometric recolouring is a technique introduced by Reinbacher et al. [72] for reclassifying misclassified geometric data. Their technique starts with a Delaunay triangulation of a bi-chromatic (red and blue) set of nodes. A node is surrounded whenever a contiguous set of oppositely-coloured neighbours (in the triangulation) spans an angle greater than 180 degrees. Their method consists of iteratively recolouring a surrounded node until no more nodes can be recoloured. In the following we formalize this idea by introducing some of the definitions and results presented by Reinbacher et al., with minor modifications.

The recolouring convention is extended to colour the links of the network as follows. Let $G = (N, L)$ be a network with bi-chromatic node set N as defined above. The links connecting either pairs of blue or red nodes are coloured blue or red, respectively. For a connected pair of differently coloured nodes, we mix the colours to obtain a *magenta link*.

Definition 9. ***(magenta angle)*** *Let the links of a bi-chromatic network $G = (N, L)$ be coloured as above. Then the* magenta angle *of a node $p \in N$ is:*

- $0°$*, if p has at most one radially consecutive incident magenta link,*
- $360°$*, if p has degree greater than one and is incident only to magenta links,*
- *the maximum angle between two or more radially consecutive incident magenta links, otherwise.*

Notice that, according to the previous definition, a node with only one neighbour in G has magenta angle $0°$ (see Figure 2.1). A surrounded node is one with magenta angle greater than $180°$.

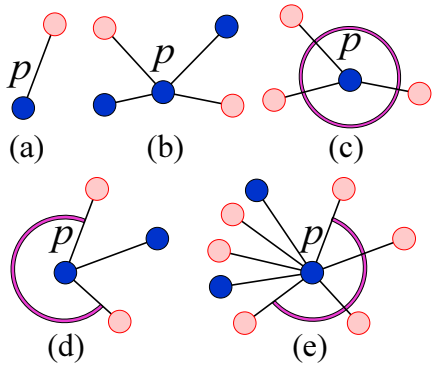

Figure 2.1: Examples of node and link colouring as well as magenta angles.

Figure 2.1 shows examples of red-blue node configurations around node p and corresponding magenta angles, α. (In a grey-scale output red nodes should appear

lighter than blue nodes.) Magenta angles larger than 0° are illustrated using arcs of circles. Thus we have angle values (a), (b) $\alpha = 0°$, (c) $\alpha = 360°$, (d), (e) $180° < \alpha < 360°$.

Definition 10. ***(opposite link)*** *We say that the link $\overline{qr}$ is an* opposite link *of $\overline{pq}$ ($p \neq r$) with respect to q if there is no link incident to q that separates $\overline{qr}$ from the ray that goes from q in the direction opposite to p (see Figure 2.2).*

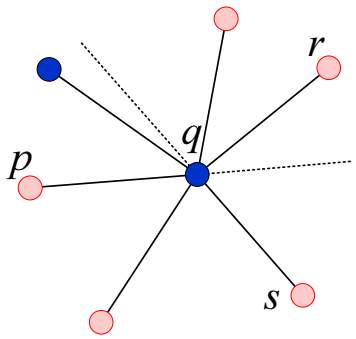

Figure 2.2: Opposite links: $\overline{qr}$ is opposite to $\overline{pq}$.

Figure 2.2 illustrates two links, $\overline{qr}$ and $\overline{qs}$, that are opposite to $\overline{pq}$ with respect to q. Notice, however, that $\overline{qp}$ is not opposite to $\overline{sq}$ with respect to q, showing the non-symmetry of the opposite relation. For simplification, we omit the "with respect to" qualifier whenever it is clear from the notation.

Lemma 1. *[72] For a link $\overline{pq}$ and any one of its opposite links with respect to q, $\overline{qr}$, if node q is recoloured, then q receives either the colour of p or the colour of r.*

Lemma 2. *[72] Let $G = (N, L)$ be a network with bi-chromatic nodes. A surrounded node q on the convex hull of N can be recoloured at most once.*

Proof. Both convex hull neighbours of q must be in the set of links that define the magenta angle of q. Thus q takes on their colour. Such neighbours of q can no longer become surrounded. This implies that q can be recoloured at most once. □

The previous lemmas have been used by Reinbacher et al. to prove the finiteness of the recolouring process in triangulations regardless of the initial colouring and the order in which the nodes are recoloured. To be more specific, they prove that no two colourings repeat, which implies a $2^n - 1$ bound on the length of recolouring sequences. It is remarkable that the recolouring order determines not only the final node colouring but also the complexity of the recolouring sequence.

Reinbacher et al. also propose a number of strategies that always produce $O(n)$ recolourings. For example, there is a 2-phase strategy that recolours at most $O(n)$ nodes: (phase 1) all the surrounded red nodes are recoloured; (phase 2) all the surrounded blue nodes are recoloured. The resulting triangulation does not accept any more recolourings since all surrounded red and blue nodes have already been recoloured. Notice that recolouring blue nodes (to red) in phase 2 does not create new red surrounded nodes. There are two problems with this approach: first, it is not "fair" in the sense that if favours blue; second, it involves the idea of phases, which cannot be implemented in a localized manner as will be needed for our fault recovery strategy. Similarly, other known strategies that guarantee a linear number of recolourings (see [72]) involve a global control that attempts against locality.

The same authors give an example (see Figure 2.3) of triangulations that may have $\Omega(n^2)$ recolourings in the worst case. The example shows three nested triangles, which can be generalized to $\Omega(n)$ nested triangles. The quadratic recolouring bound is attained whenever surrounded nodes on nested triangles as close to the centre as possible are recoloured first. Reinbacher et al. proposed as an open problem closing the gap between the exponential upper bound mentioned above and the quadratic lower bound provided by this example.

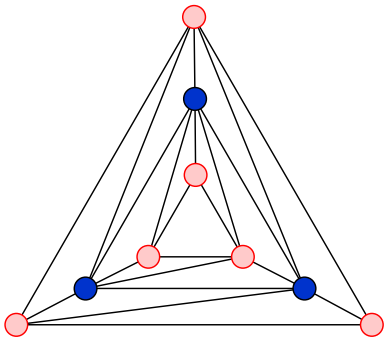

Figure 2.3: (Adapted from [72]) Nested triangles example in which $O(n^2)$ recolourings may occur.

2.3.2 Geometric Recolouring of Triangulations

In the following we show the derivation of a tight bound for the length of recolouring sequences in triangulations. We start with some additional definitions.

Definition 11. ***(extremal magenta links)*** *Let q be a surrounded node at the beginning of iteration j of the recolouring sequence, then there exists a maximal consecutive sequence of magenta links incident to q which we denote by $C_q(j) = (\overline{qp_1}, \overline{qp_2}, \ldots, \overline{qp_k})$. We say that the links $\overline{qp_1}$ and $\overline{qp_k}$ are the* extremal magenta links *in $C_q(j)$.*

Lemma 3. *Let q be a surrounded node (magenta angle ¿ 180°) at the beginning of iteration j, such that $\overline{pq}$ is extremal in $C_q(j)$ and $\overline{qr}$ is any opposite link of $\overline{pq}$. Then both p and r are the same colour, that is not the colour of q.*

Consider a network $T = (N, L)$, a triangulation of an arbitrary node set N. We define an *internal node* as a node that is not on the boundary of the convex hull of T. Similarly, an *internal link* is a link incident to at least one internal node. Note that the convexity of the faces in the triangulation ensures that for any internal link $\overline{pq}$ that is incident to an internal node q there always exists at least one opposite link with respect to q, $\overline{qr}$, such that the nodes p, q, r appear in x-coordinate order.

Definition 12. ***(opposite chain and monotone chain)*** *Let G be a network. An* opposite chain *is a simple path $P = (p_0, \ldots, p_m)$ in G such that*

- $p_0 \neq p_m$
- *either $\overline{p_i p_{i+1}}$ is opposite to $\overline{p_{i-1} p_i}$ or $\overline{p_i p_{i-1}}$ is opposite to $\overline{p_{i+1} p_i}$, for all $1 < i < m - 1$.*

If P is monotone with respect to the x-axis coordinate, the chain is denoted a monotone chain.

Definition 13. ***(colour change number)*** *Let $P = (p_0, \ldots, p_m)$ be an opposite chain. The* colour-change number *of the chain P, $\chi(P)$, is the number of magenta links in P. Similarly, the colour-change number of a set of monotone chains C, $\chi(C)$, is defined as the total colour-change number over all chains in C.*

We can cover T using a set C of monotone chains as follows. For every link $\overline{pq}$ in T we can obtain a monotone chain $P_{\overline{pq}}$ starting with link $\overline{pq}$, then adding links $\overline{rp}$ and $\overline{qs}$, opposite to $\overline{pq}$. This process is repeated iteratively by inserting opposite links in both directions from $\overline{pq}$ until the nodes with smallest and largest x-coordinates are reached. These are obviously nodes on the convex hull of T. See Figure 2.4 for an example. Planarity implies a total of $O(n)$ links in T and therefore, $O(n)$ chains in C. A similar idea of covering a planar subdivision with monotone chains has been used by Lee and Preparata [45] with a different purpose.

Theorem 4. *Given a set of n bi-chromatic nodes, N, together with a triangulation $T = (N, L)$ of N, any recolouring sequence of T consists of at most $O(n^2)$ recolourings.*

Proof. Consider a monotone chain cover C for the set of links of T, and let P be a monotone chain in C. We observe how the colour-change number of C changes with

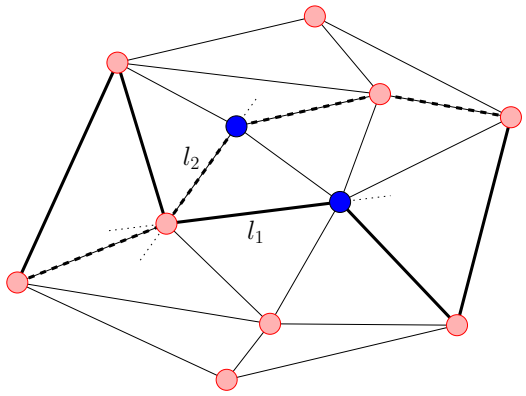

Figure 2.4: Monotone chains corresponding to the links l_1 (thick lines) and l_2 (thick dashed lines)

every node recolouring. The analysis is divided into recolourings that occur at the convex hull node, and recolourings that occur at internal nodes of T. From Lemma 2 it follows that convex hull recolourings can occur at most once per convex hull node. Lemma 1 implies that the colour change number $\chi(P)$ cannot increase when any internal node is recoloured.

We will see that for every recolouring of an internal node, there is always at least one monotone chain whose colour-change number decreases. Suppose that at some step j of the recolouring sequence the internal node q changes colour. If the magenta angle of q is less than 360° we take an extremal magenta link $\overline{pq}$ in $C_q(j)$; otherwise all links incident to q are magenta and we choose $\overline{pq}$ arbitrarily. An opposite link of $\overline{pq}$, $\overline{qr}$, is in the monotone chain $P_{\overline{pq}}$. Furthermore, node r must be the same colour as p and different from q, by Lemma 3. Thus, the colour-change number of $P_{\overline{pq}}$ must decrease by two. See Figure 2.5.

Obviously, the colour-change number of any monotone chain is at most $n-1$ and can increase (by one) only twice at its endpoints (nodes on the convex hull with minimum and maximum x-coordinates) during the entire recolouring sequence. Since

the number of chains in C is $O(n)$, $\chi(C)$ is $O(n^2)$. This number is not significantly affected (asymptotically) by the possible linear increase of the colour-change number of C at the nodes of minimum and maximum x-coordinates. On the other hand, every recolouring of an internal node q produces a decrease in $\chi(P_{\overline{pq}})$ for at least one chain $P_{\overline{pq}} \in C$, while the colour-change number for all other chains in C either decreases or remains unchanged. Thus, $\chi(C)$ decreases (by at least two) with the recolouring of an internal node. This proves that at most $O(n^2)$ internal node recolourings can occur. This together with the $O(n)$ number of convex hull node recolourings add up to $O(n^2)$ recolourings, which completes our proof. □

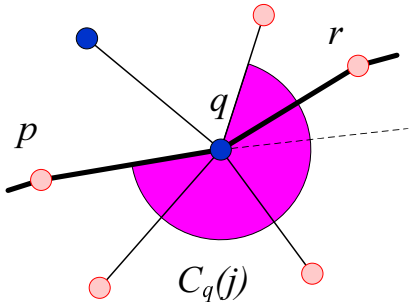

Figure 2.5: The colour change number of a monotone chain $\chi(P_{\overline{pq}})$ decreases when q is recoloured.

2.3.3 Geometric Recolouring of Degree-3 Networks

In this section we consider geometric recolouring on networks with maximum node degree three. At all times the networks are assumed to be connected because, in general, each connected component can be considered independently. Next we prove a linear bound for these networks.

Theorem 5. *Let $G = (N, L)$ be a network with n bi-chromatic nodes. If G has maximum node degree 3, the length of any recolouring sequence of G is $O(n)$.*

Proof. The proof follows from an observation on how the number of magenta links decreases with each recolouring. Let p be a node that is being recoloured. At least two links incident to p need to be magenta, according to the definition of magenta angle. These links change to a *solid* (red or blue) colour after the recolouring of p. Before p is recoloured, at most one link of solid colour can be incident to it, given that $deg(p) \leq 3$. This link, if it exists, becomes magenta. Therefore, the number of magenta links decreases by at least one with each recolouring. As the initial number of magenta links is $O(n)$, the number of recolourings is also $O(n)$. □

The proof of Theorem 5 relies only on the degree of a node. Thus, this bound also holds for networks with link crossings as long as they have maximum degree 3. Also, in Section 2.3.7 we further extend the scope of this theorem to include non-straight link networks.

One may ask whether a recolouring sequence recolours every node at least once. This question is of particular interest for the case of infinite recolouring sequences in planar and non-planar graphs (see Subsection 2.3.5). The following proposition answers this question in the negative sense for networks with maximum node degree three.

Proposition 6. *Let D be a network with n bi-chromatic nodes. If D has maximum node degree 3, then no recolouring sequence of D recolours all nodes.*

Proof. Recall that a node with degree 0 or 1 never gets recoloured. Thus, we assume that all nodes have degree 2 or 3, or else the theorem is trivially true.

We continue by orienting the links. For a node p of degree 2 we orient both incident links toward p. If the maximum angle defined by the links incident to a node p of degree 3 is less than or equal to 180° then we orient all 3 links toward p. Finally,

for the case where the maximum angle defined by two adjacent links at p is greater than 180° we orient the two links toward p and the third link away from p. Observe that with this orientation scheme, if a node p is surrounded, then its incoming links must be magenta. Furthermore, using this orientation scheme and the fact that a majority of the links are oriented inward, the pigeon-hole principle implies that there is at least one links, $\overline{pq}$, oriented inward at both endpoints p and q. Putting these two observations together we can conclude that in any recolouring scheme only one of p or q can ever be recoloured, thus showing that no recolouring scheme recolours all nodes. □

2.3.4 Geometric Recolouring of Plane Networks

One may think of obtaining recolouring bounds for plane networks (i.e., networks embedded in the plane in which no two links cross) based on the fact that a plane network is a subgraph of a triangulation. However, this is not the case. There are simple examples of networks G, and $S \subset G$, such that S has either a larger or smaller number of recolourings in the worst case over all initial colourings.

In fact, a recolouring sequence of a plane network can be infinite. Figure 2.6 shows an example of a plane network and a colour configuration that may lead to an infinite recolouring sequence (see Appendix A.1, Figures A.1 and A.2 for a sequence that repeats a colour configuration). In the figure the nodes represented by small disks never change colour and the nodes represented by large disks can change colour an infinite number of times.

By carefully adding more links incident to the nodes of degree one in this example (Figure 2.6), the network can be made 2-connected, and its minimum node degree

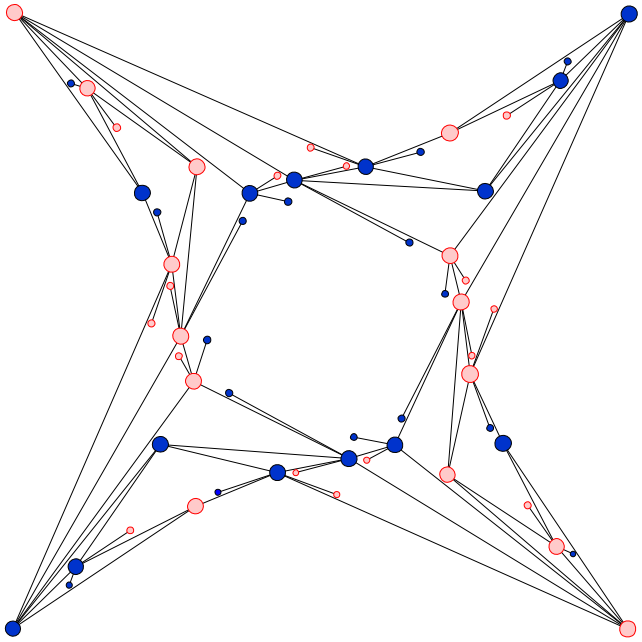

Figure 2.6: 60-node network that has an infinite recolouring sequence.

increased, without affecting the recolouring sequence. This example can also be augmented by attaching other nodes and links on the "outside" of any reflex node on the outer face without affecting the infinite recolouring sequence. The recolouring lower bound for plane networks is generalized in the following theorem.

Theorem 7. *There exist bi-chromatic networks with 60 or more nodes that have infinite recolouring sequences.*

Our example in Figure 2.6 was drawn for clarity of exposition. In fact, we have examples of planar graphs with smaller number of nodes that have infinite recolouring sequences. For instance, a 48 nodes plane network with infinite recolouring sequence can be obtained from the example in Figure 2.6 by carefully merging pairs of nodes that lie on the same face and never change colour.

There are, however, classes of plane networks, other than triangulations, for which

the recolouring sequences are provably finite. One such example is implied by a more general result as stated in the next section (see Theorem 10).

2.3.5 Geometric Recolouring of Not-Necessarily-Plane Networks

At this point, it is obvious that one can also find non-plane networks with infinite recolouring sequences since plane networks allow so. Nevertheless, the examples shown for infinite recolouring sequences on plane networks include nodes that never change colour. In Figure 2.7 we show an example of a non-plane network that has an infinite recolouring sequence in which every node changes colour infinitely many times (see Appendix A.1, Figure A.3 for an example infinite recolouring sequence). If similar examples can be built for plane networks, these have not yet been found.

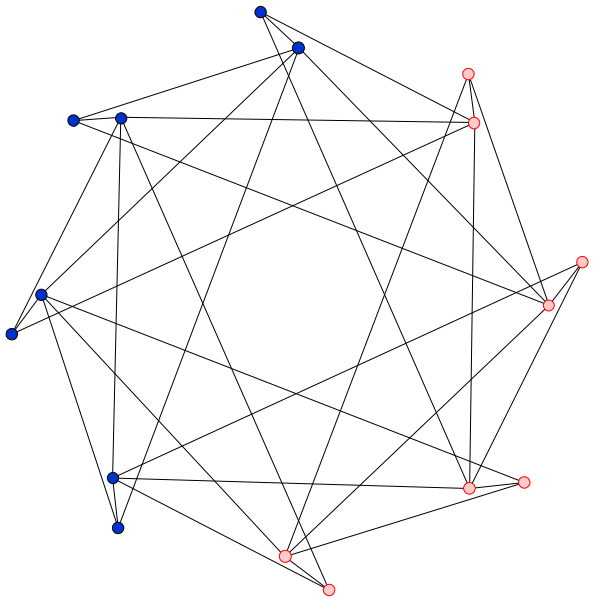

Figure 2.7: Non-plane network with bi-chromatic nodes that has an infinite recolouring sequence.

Notice that the example shown in Figure 2.7 can be augmented by attaching more nodes and links at any of the existing nodes towards the "inside" of any of the acute angles, without affecting the infinite recolouring sequence.

Theorem 8. *There exist bi-chromatic networks with 16 or more nodes that have infinite recolouring sequences in which every node changes colour infinitely many times.*

The infinite recolouring example shown in Figure 2.7 is not minimal: a 10-node network with infinite recolouring sequence can be obtained from the example in the figure if only 5 pairs of nodes are used, instead of 8, and links are slightly changed. For clarity, we do not show a smaller example.

There are also families of networks where recolourings always end after a finite number of steps. One such class has already been characterized in Theorem 5. Another class is formally described in what follows.

Definition 14. ***(convex node and convex links)*** *A* convex node *p of a network is a node with two consecutive incident links, the* convex links *with respect to p, that define an angle greater than* $180°$.

Lemma 9. *Let p and q be two convex nodes that share a convex link $\overline{pq}$. Link $\overline{pq}$ cannot change from red or blue to magenta.*

Proof. Figure 2.8 depicts the only two possible scenarios that satisfy the hypothesis of the lemma. It is obvious that a span of an angle greater than $180°$ around p or q needs to include link $\overline{pq}$, given the hypothesis of the lemma. Therefore, if $\overline{pq}$ is red or blue, neither p nor q can be surrounded. □

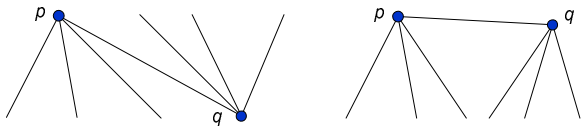

Figure 2.8: Two cases of adjacent convex nodes p and q sharing a convex link.

Theorem 10. *Let $G = (N, L)$ be a (not necessarily plane) network with set of bichromatic nodes N and set of links L, such that every node p in G satisfies one of the following three conditions:*

- *p has degree less than or equal to 1,*
- *p is not convex,*
- *p is convex and is adjacent to another convex node through a convex link (i.e., p is not an* isolated convex *node).*

The length of recolouring sequences of G is $O(|N||L|)$.

Proof. We use an opposite chain cover C of the network, such that C contains an opposite chain $P_{\overline{pq}}$ for every link $\overline{pq}$ in G. Recall that, by Lemma 1, $\chi(P_{\overline{pq}})$ cannot increase with the recolouring of an internal node of $P_{\overline{pq}}$. The chains in C are constructed in a way that monotonicity is maintained (along the x-axis coordinate) whenever possible. Initially, any opposite chain $P_{\overline{pq}}$ consists of the single link $\overline{pq}$. Then $P_{\overline{pq}}$ is extended in both directions by addition of opposite links. This way $P_{\overline{pq}}$ can be extended (in each direction) until one of the following conditions is met:

- A node of degree one is reached.
- A convex node is reached.

The first case is benign since, by definition, a node of degree one is never surrounded and the colour-change number of an opposite chain can not increase at this node.

For the second case we show how to adjust $P_{\overline{pq}}$ such that $\chi(P_{\overline{pq}})$ can increase at most by one per endpoint. Let $P_{\overline{pq}} = (p_0, \ldots, p_m)$ and let p_0 be a convex node –the following analysis symmetrically applies to p_m. By the conditions of the theorem, there exists a convex node p adjacent to p_0 such that $\overline{p_0 p}$ is a convex link. If $p \neq p_1$, we add p to P such that $P = (p, p_0, \ldots, p_m)$; otherwise, P is left unchanged. In either case the first two nodes of P, let them be p and p_0, or p_0 and p_1, form a pair of convex nodes connected by a convex link. Thus, by Lemma 9, the first node of P can change colour at most once. Therefore, $\chi(P_{\overline{pq}})$ can increase at most once (by one) at each endpoint.

We have shown that the colour-change number of a monotone chain does not increase significantly. On the other hand, there is always at least one monotone chain whose colour-change number decreases. Suppose that at some step j of the recolouring sequence the internal node q changes colour. If the magenta angle of q is less than 360° we take an extremal magenta link $\overline{pq}$ in $C_q(j)$; otherwise all links incident to q are magenta and we choose $\overline{pq}$ arbitrarily. An opposite link of $\overline{pq}$, $\overline{qr}$, is in the monotone chain $P_{\overline{pq}}$. Furthermore, by Lemma 3, node r must be the same colour as p and different from q. Thus, $\chi(P_{\overline{pq}})$ must decrease by two. (See Figure 2.5.)

The colour-change number of any opposite chain in C is at most $|N| - 1$ at all times, because the construction of the chain guarantees that no node is visited twice. Notice that the colour-change number of a chain can increase (by one) only

twice at its endpoints during the entire recolouring sequence. Since the number of chains in C is $O(|L|)$, $\chi(C)$ is $O(|N||L|)$. This number is not significantly affected (asymptotically) by the possibly $O(|L|)$ increase of the colour-change number of C at the opposite chain endpoints. On the other hand, every recolouring of an internal node q produces a decrease in $\chi(P_{\overline{pq}})$ for at least one chain $P_{\overline{pq}} \in C$, while the colour-change number for all other chains in C either decreases or remains unchanged. Thus, $\chi(C)$ decreases (by at least two) with the recolouring of an internal node. This proves that at most $O(|N||L|)$ internal node recolourings can occur, which together with the $O(|L|)$ number of opposite chain endpoint recolourings produces the intended bound of $O(|N||L|)$ recolourings overall. □

In the following, we refer to the conditions stated in this theorem as the conditions of Theorem 10 or the non-isolated-convex (NIC) conditions. By extension, we define a NIC network as follows.

Definition 15. ***(NIC network)*** *Let G be a (not necessarily plane) network. G is a* NIC network *if it satisfies the conditions of Theorem 10, that is, the NIC conditions.*

For completeness we show that when the network satisfies the NIC conditions, not all nodes are recoloured along the same recolouring sequence.

Theorem 11. *Let $G = (N, L)$ be a NIC network with n bi-chromatic nodes. No recolouring sequence of D recolours all nodes.*

Proof. Obviously G is finite, and therefore bounded. We assume there is no node of degree one otherwise the theorem trivially holds. Thus, there exists at least one convex node. By the hypothesis of the theorem, such convex node is adjacent to another convex node through a convex link. Lemma 9 shows that two convex nodes

depend on each other for recolouring: if one of the nodes changes colour, the other one does not. Thus, one can always find a node that does not change colour. □

Theorem 12. *Let $T = (N, L_T)$ be a triangulation with n nodes and let $G = (N, L)$ be a network with the same node set as T and $L_T \subset L$. The recolouring sequences of G have $O(n|L|)$ length.*

Proof. It is known from Theorem 4 that all recolouring sequences of T have $O(n^2)$ length and that this can be proved by using a monotone chain cover. Since $L_T \subset L$, it is also possible to find a monotone chain cover C of G. The reason is that L also contains opposite links that maintain the monotonicity of the opposite chains at all times. Each monotone chain has $O(n)$ length. The $O(|L|)$ links of G are assigned one monotone chain each. Therefore, a monotone chain cover of G has an overall $O(n|L|)$ complexity and $\chi(C)$ is also $O(n|L|)$. Based on this fact, we conclude that any recolouring sequence of G has $O(n|L|)$ recolourings as argued in Theorem 4. □

In a similar way one can prove that plane networks where every internal face is convex have a quadratic number of recolourings and that any supergraph of such convex-faced network has also a finitely (at most cubic) long recolouring sequences. Next we study recolouring sequences in trees.

2.3.6 Geometric Recolouring of Trees

In this subsection we use the term *tree network* to refer to a connected acyclic network (not necessarily planar). A trivial example of a tree network that has a $O(n)$ recolouring sequence is a "jigsaw" path with nodes alternately coloured. In such an example, all blue nodes can be recoloured to red, leading to approximately $n/2$ recolourings.

An argument similar to the proof of Theorem 10, using opposite chains, can be made to prove a quadratic upper bound on the number of recolourings of tree networks. However, this bound is not tight. As a corollary of Theorem 5 it follows that binary tree networks, i.e., tree networks with maximum node degree 3, have a linear number of recolourings. In this section we prove that the number of recolourings of tree networks in general is also linear.

In order to prove a tight bound (Theorem 16) on the number of recolourings of tree networks, we define a partial order on the recolourings involved in a recolouring sequence. We then bound the total number of recolourings based on the number of minimal elements (sinks) of such a partial order. This idea is explained and formalized in the remainder of this section.

We denote a recolouring event, or simply a recolouring, r as the event of a certain node p being recoloured. Let $R = (r_1, \ldots, r_k)$ be a recolouring sequence in which r_i denotes the recolouring at step i, $1 \leq i \leq k$, $k > 0$. We also denote $p(r_i)$ as the node that changes colour at recolouring r_i, and $N(r_i)$ the number of times that $p(r_i)$ has changed colour in R prior to event r_i.

Definition 16. ***(history graph)*** *Let $T = (N, L)$ be a tree network and let R be a recolouring sequence of T. The* history graph *of R is a directed graph $H = (R, I)$, $I \subset R \times R$ such that $(r_j, r_i) \in I$ if and only if there exists $r \in R$, such that $\overline{p(r)p(r_j)} \in C_{p(r_j)}(j)$, and $i = max(\{l : 1 \leq l < j, p(r_l) = p(r)\})$.*

In plain words, the history graph of a given recolouring sequence contains all the recolourings as vertices and certain dependencies among the recolourings as edges. More specifically, there exists an edge from recolouring r_j to recolouring r_i if and only if r_i is the last recolouring of a neighbour $p(r_i)$ of $p(r_j)$ prior to r_j, such that

$p(r_i)$ is in the magenta angle associated to r_j. See Figure 2.9 for an example. Notice that, according to the previous definition, for any two recolourings, r_i, r_j, such that $(r_k, r_i) \in I$ and $(r_k, r_j) \in I$, $p(r_i) \neq p(r_j)$.

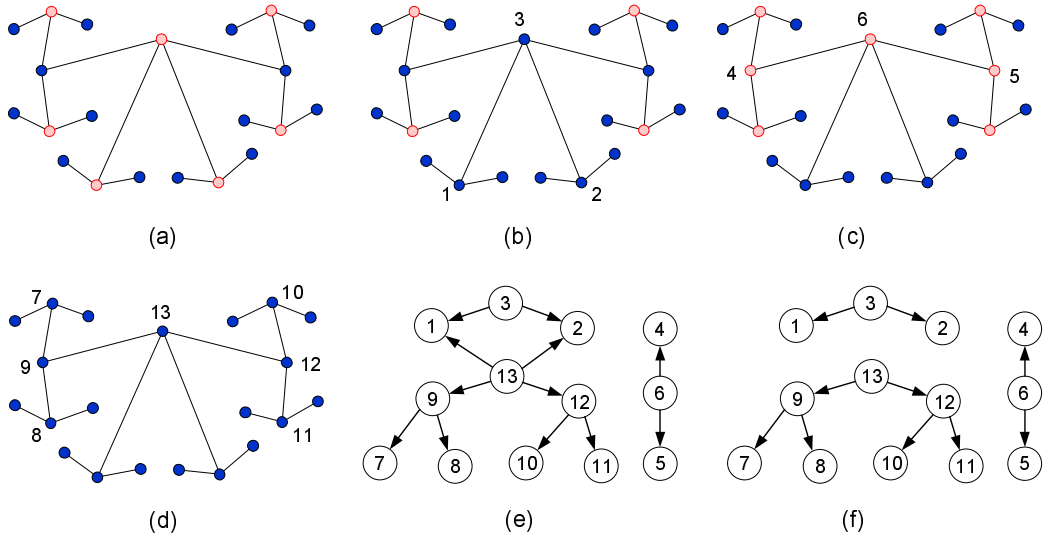

Figure 2.9: Example tree recolouring with associated structures (history graph and binary history graph).

The example in Figure 2.9 shows a tree (a) with its recolouring sequence $R = \{r_1, \ldots, r_{13}\}$ at different stages (b, c, d). The indices of the recolourings appear as labels at the corresponding nodes. The history graph (e) and binary history graph (f) of R are also shown.

Observation 13. *By the definition of history graph all the edges are directed from later recolourings to earlier ones. Therefore, a history graph is a directed acyclic graph (DAG) and defines a partial order on the elements of the recolouring sequence.*

The following lemma proves that there are at least two neighbours of a node p that are recoloured at least once between two consecutive recolourings of p.

Lemma 14. *Let T be a tree network with n bi-chromatic nodes, let R be a recolouring sequence of T, and let $H = (R, I)$ be the history graph of R. Consider a recolouring $r \in R$ with $N(r) > 0$. Then the outdegree of r is at least 2.*

Proof. Obviously, if a node is recoloured red (or similarly blue) and was recoloured earlier in the sequence, the previous recolouring was to blue (red). The intersection between the magenta angles at the time it is surrounded by red (blue) and previously by blue (red) contains at least two links because the corresponding magenta angles are greater than 180°. Therefore, there are at least two neighbours of p, $p_1, p_2 \in T$, that are recoloured at least once between two consecutive recolourings of p. □

In the light of Lemma 14, we can state the following definition.

Definition 17. ***(binary history graph)*** *Let R be a recolouring sequence of a tree network T and $H = (R, I)$ be the corresponding history graph. The* binary history graph *of R, $BH = (R, BI)$, $BI \subseteq I$, is a subgraph of the history graph where vertices have outdegrees 2 or 0: vertices with outdegree 0 correspond to first time recolourings; vertices with outdegree 2 correspond to subsequent recolourings. Consider a node r_k such that $N(r_k) > 0$, that is, a recolouring of a node that has been previously recoloured. We choose the two outgoing edges of r_k, (r_k, r_i), (r_k, r_j), such that i and j are the largest indices smaller than k for neighbours of r_k in the history graph.*

The motivation to define the binary history graph is to obtain a cycle free subgraph of the history graph that involves all the recolourings (see Figure 2.9 (f)). In the next lemma we prove that the binary history graph is, in fact, cycle free.

Lemma 15. *Let T be a tree network, and R a recolouring sequence of T with binary history graph BH. BH has no directed or undirected simple cycles. Therefore, BH*

is a forest of trees.

Proof. Since BH is a subgraph of the history graph of R, BH is also a DAG, by Observation 13. Therefore, there are no directed cycles in BH. Any undirected simple cycle BH would have at least one vertex r with two outgoing edges and one vertex s with two incoming edges. For the sake of contradiction, we assume that there exists such a simple cycle, C, in BH.

Consider the function $f : R^* \rightarrow V(T)^*$ such that $f(r_1, r_2, \ldots, r_k) = p(r_1), p(r_2), \ldots, p(r_k)$, $r_i \in R, 1 \leq i \leq k$. In particular, f maps a path in BH to a path in T. Let P_1 and P_2 be the two undirected paths that connect r and s in C. By the definition of binary history graph, the outgoing edges of r are incident to vertices t_1 and t_2 such that $p(t_1) \neq p(t_2)$. Thus, $|C| > 2$. Without loss of generality, let t_1 be in P_1 and t_2 be in P_2. Then paths $f(P_1)$ and $f(P_2)$ are different at nodes $p(t_1)$ and $p(t_2)$. This implies that there are two different paths in T connecting $p(r)$ and $p(s)$. Thus, we establish a contradiction. □

To obtain a bound on the size of binary history graphs we show that the number of nodes is linear in the size of the corresponding tree network. This will lead us to conclude the results of the following theorem.

Theorem 16. *Let $T = (N, L)$ be a tree network with n bi-chromatic nodes. The length of any recolouring sequence of T is $O(n)$.*

Proof. Let R be any recolouring sequence of T, BH the binary history graph of R, and V and E the set of vertices and edges of BH, respectively. In order to prove this theorem we show that $|V| = |R|$ is $O(n)$. Let V_k denote the set of vertices of degree k in V, and V_{k^+} the set of vertices of degree at least k in V. For accounting purposes, we partition the set of vertices of BH into four classes: V_0, V_1, V_2, and V_{3^+}.

Vertices of degree 0 and 1 are all first-time recolourings (sinks) according to the definition of binary history graph, because they have 0 outgoing edges. Also, every vertex of degree 2 is either a sink or a source because internal vertices have degree at least 3, that is, one or more incoming edges and two outgoing edges. We partition the set of degree-2 vertices into V_2^t, the sinks of degree 2, and V_2^s, the sources.

Next we show that the binary history graph has less sources than non-source vertices (i.e., $|V_2^s| < |V \setminus V_2^s|$). Suppose, for the sake of contradiction, that $|V_2^s| > |V \setminus V_2^s|$). Let $k = |V_2^s|$ be the number of sources. Because there are 2 distinct edges incident to each source, the overall number of edges in BH satisfies $|E| \geq 2k$. Given that BH is a forest of trees, $|E| = |V| - m$, where $m > 0$ is the number of connected components of BH. It follows that $|V| > 2k$. Therefore, $|V \setminus V_2^s| > k = |V_2^s|$, which contradicts the original assumption. We use the new bound on the number of sources to derive the following equations.

$$|V| = |V_0| + |V_1| + |V_2^s| + |V_2^t| + |V_{3^+}| \leq 2(|V_0| + |V_1| + |V_2^t| + |V_{3^+}|). \tag{2.1}$$

Notice that at most n vertices can be sinks since in the worst case all nodes of T are recoloured for the first time. Consequently,

$$|V_0| + |V_1| + |V_2^t| \leq n. \tag{2.2}$$

Thus, a linear bound on $|V_{3^+}|$ entails a linear bound on $|V|$. We derive such bound in what follows. From properties of graphs and, in particular, of forests of trees, and from Equation (2.1),

$$\sum_{r \in V} deg(r) = 2|E| < 2|V| - 2m < 2(|V_0| + |V_1| + |V_2| + |V_{3^+}|), \tag{2.3}$$

According to the definitions of V_k and V_{k^+},

$$\sum_{r \in V} deg(r) \geq |V_1| + 2|V_2| + 3|V_{3^+}|. \tag{2.4}$$

Equations (2.3) and (2.4) lead to

$$|V_{3^+}| \leq 2|V_0| + |V_1| - 2m. \tag{2.5}$$

Observe that the number of vertices of degree 0 cannot be greater than the number of connected components. Thus, it follows that $|V_0| \leq m$ and $|V_{3^+}| \leq |V_1|$. Combining this with (2.2) we obtain

$$|V_{3^+}| \leq n. \tag{2.6}$$

Finally, from (2.1), (2.2), and (2.6) we have

$$|V| \leq 2(|V_0| + |V_1| + |V_2^t| + |V_{3^+}|) \leq 4n \tag{2.7}$$

Thus, we conclude that the length of any recolouring sequence of T is at most $4n$, or $O(n)$. □

2.3.7 Extensions

Non-Straight Links

Recolourings may also occur in non-straight link embeddings of networks –wired networks, for instance. One can consider a node as surrounded whenever the magenta angle defined by the tangents of a set of consecutive magenta links leaving the node is greater than 180°. Theorem 5 also holds in this case, that is, any non-straight network with minimum node degree 3 has a recolouring sequence of length at most $O(n)$. However, for more general networks the results seem to differ from straight-link

network. Figure A.4 in Appendix A.1 shows a simple example of a plane network where the links have at most one bend and there is a recolouring sequence in which all the nodes change colour infinitely many times.

More Than Two Colours

Suppose that the nodes can be of more than two categories or colours. We define the colour of a link as the mixture of the colours of its incident nodes. In a multicoloured scenario we say that p is *surrounded* by a set of links of a single mixed colour if the links define a continuous angle greater than 180°. As we may intuitively observe, increasing the number of colours only lowers the chances of a node being surrounded without changing the fundamental nature of the problem. In fact, inspection shows that all of our previous definitions and results hold in a multicoloured scenario. Thus, our recolouring bounds for a bi-chromatic set of nodes carries over to multicoloured node sets.

Threshold Angles Greater Than 180°

So far we have considered a node as surrounded whenever its magenta (or single mixed colour) angle is greater than 180°. We call this the recolouring threshold angle and denote it by θ. For many applications it is not convenient to set a value for θ that is smaller than 180° because trivial recolouring cycles may exist, leading to infinite recolouring sequences (see [72]). However, one may want to consider recoloured nodes when $\theta > 180°$. It is then an interesting question whether the bounds previously presented hold for any $\theta > 180°$.

Notice that on the plane network example with infinite recolouring sequences there

are surrounded nodes where the magenta angle is close to 180°. It is unknown what the largest θ value is so that infinite recolouring sequences exist in plane networks. For triangulations, one can adapt the example shown in Figure 2.3 in a way that the nested equilateral triangles are as close to the each other as desired, allowing quadratic recolouring sequences for any given $\theta < 300°$. It is not known, however, whether example triangulations that allow quadratic recolouring sequences exist for $\theta \geq 300°$. We conjecture that for any $\theta \geq 300°$ triangulations allow at most a linear number of recolourings.

2.4 Fault Recovery in Wireless Networks

Before presenting the application of geometric recolouring to the fault recovery problem we propose a general framework for recolouring in wireless network. Within this framework we propose a combinatorial approach to recolouring, that has been previously used for fault recovery. We then present our geometric approach along with a set of challenges and proposed solutions. We also propose a hybrid method that combines the combinatorial and geometric approaches. Finally, experimental results are presented to compare the different methods.

2.4.1 A General Fault Recovery Algorithm

We examine different criteria for a node to be considered surrounded or dominated by neighbours of the opposite colour. For now, we assume that we are provided with a general function SURROUNDED that returns TRUE for a node if it is surrounded

by nodes of the opposite colour, or FALSE otherwise. Different instances of SURROUNDED will be studied below.

As pointed out earlier, a recolouring does not necessarily mean that a faulty node is being fixed; on the contrary, sometimes a healthy node can become faulty by the same mechanism. However, our experiments demonstrate that whenever the cause of the fault affects only a relatively small number of nodes, the number of nodes that are recovered from faults is substantially larger than the number of nodes that turn faulty. In fact, in many cases all faulty nodes are able to recover (see Section 2.4.5 for more details).

The fault recovery algorithm simply consists of the *Recolouring Protocol*, as defined next, to be executed at all nodes. The protocol defines successive recolourings of a node, according to the function SURROUNDED, and a mechanism to notify its neighbours whenever a change of colour occurs.

Step 3 of the recolouring protocol ensures that the neighbours of a node become aware of its colour changes. The parameter T can be adjusted for an optimal tradeoff between fast fault recovery and low network communication. The algorithm terminates (temporarily) whenever no more nodes can be recoloured. However, it restarts itself within T time units of the occurrence of a new fault.

Notice that the nodes do not necessarily wait until they know the colour of all neighbours. The protocol is presented in a way that no synchrony restriction is imposed. This will not affect the termination of the algorithm, as will become evident from the results presented in the next two sections; but, it may affect the resulting number of faulty nodes. One can assume, however, that no faults will occur immediately after nodes are deployed, but some time after the protocol has started.

Algorithm 1: Recolouring Algorithm

input : Network $G = (N, L)$ with bi-chromatic nodes.
output: Network $G = (N, L)$ with a different node colouring such that no more nodes can be recoloured.

Recolouring Protocol

Step 1. Broadcast a COLOUR message to all neighbours, with the node's colour information.

Step 2. If there is a COLOUR message in the node's message queue, the neighbour's colour information is updated and the colour of the node itself is updated according to the return value of the SURROUNDED function.

Step 2.1. If the colour changes, send a message to all neighbours with the new colour information.

Step 3. If there is no COLOUR message in the queue and no COLOUR message has been received for T time units, go to Step 1.

Step 4. Go to Step 2.

The idea that this algorithm succeeds in a completely asynchronous setting is also a consequence of the choice of surrounded criteria. If the specific surrounded criterion guarantees that the process terminates when recolourings occur sequentially, it will also guarantee termination with high probability in a purely asynchronous environment. The reason is that in an asynchronous environment two recolourings occur simultaneously with a certain probability, smaller than one. Thus, under the reasonable assumption that simultaneous recolouring are independent from one another, the joint probability for a sequence of simultaneous recolourings tends to zero as the number of considered recolourings tends to infinity.

If, on the other hand, recolourings occur synchronously for all nodes, such that all nodes that can be recoloured at each round change their colour, then the process may not terminate. Goles and Olivos [32] explain the behaviour of such systems and how they fall into configurations that oscillate infinitely. In fact, they proved that for certain strategies, including the geometric approach as explained in the next section, the system oscillates with periodicity two.

We define the *sequential model* as the model in which no simultaneous recolourings occur, as if there was a global scheduler controlling the network's activity. In the following we limit our study of recolouring strategies to the sequential model.

2.4.2 The combinatorial approach

The SURROUNDED function can be implemented based on simple combinatorial properties of a node and its neighbours. In this case, a majority rule is considered. Thus, we define a function SURROUNDED_COMBINATORIAL that returns TRUE for nodes with more neighbours of the opposite colour than neighbours of its colour and FALSE

otherwise. The majority and other counting rules, because of their simplicity, are applied in a wide variety of dynamic systems of this kind: they are at the core of cellular automata theory (see Clarridge's thesis for definitions and applications to wireless sensor networks [19]); and are also the strategies used for dealing with faults by other authors [21, 44, 48].

In order to prove that this simple strategy terminates, we extend our colouring convention to colour the links of the network as in the previous section. That is, we have red, blue, and magenta links.

Theorem 17. *Let $G = (N, L)$ be a network with bi-chromatic node set N. In the sequential model, the Recolouring Algorithm with parameter T and the* SURROUNDED_COMBINATORIAL *function terminates after $O(|L|)$ recolourings from the time of the last fault plus T.*

Proof. We use a simple counting argument on the number of magenta links. The number of magenta links is obviously at most $|L|$. After T units of time from the last fault, any notification of colour change (i.e., COLOUR message) is a consequence of a recolouring, as opposed to a delayed broadcast from a node that has become faulty due to environmental factors. Then, with every recolouring, the number of magenta links incident to the recoloured node decreases. Because no other recolourings occur simultaneously, according to our definition of the sequential model, the overall number of magenta links decreases with every recolouring. Thus, at most $O(|L|)$ recolourings can occur, which proves the theorem. □

2.4.3 The geometric approach

The geometric-based criterion used for fault recovery in wireless networks is similar to the strategy reviewed in Section 2.3 for reclassification of geographic data. We assume that each node knows all its neighbours and the angle each neighbour is from it. The angle is taken clockwise with respect to the North direction. This is not a very demanding assumption as the angle information can be obtained by unexpensive devices such as arrays of radio antennas [55] or ultrasound receivers [71], for instance. In fact, several authors have used the angle information for their algorithms [43, 12], and this is obviously less demanding than knowing the exact location of all neighbours, which is also a common assumption (see [10] for example). With the angle information at hand, a new technique can be developed for implementing the SURROUNDED function of Algorithm 1. In fact, the geometric criterion we use for fault recovery is geometric recolouring as presented in the previous section. That is, the function SURROUNDED_GEOMETRIC is defined to return TRUE if the magenta angle of the node in question is greater than 180°, and FALSE otherwise. However, in order to guarantee termination, some preprocessing of the network is required. In what follows we provide a set of preliminary results required to introduce the preprocessing algorithm and the geometric approach to fault recovery.

Preprocessing the network

It is known that a geometric recolouring process can be infinite for general (non-planar) networks. Figure 2.7 shows an example network with an infinite recolouring sequence. Because we would like the fault recovery process to terminate, the recolouring strategy cannot be directly applied to the original network; instead a network is

computed to represent the topology of the original network as best as possible while having finite recolouring. To this end we use NIC networks as introduced in the previous section (see Definition 15). One of the advantages of using NIC networks is that they can be described using only local properties of the nodes and therefore, can be computed locally at the nodes.

Corollary 18. *(of Theorem 10)*
Let $G = (N, L)$ be a NIC network with bi-chromatic node set N. In the sequential model, the Recolouring Algorithm with parameter T and the SURROUNDED_GEOMETRIC *function terminates after $O(|N||L|)$ recolourings from the time of the last fault plus T.*

Corollary 18 makes it clear that NIC networks can be used for finite recolouring. In what follows, we show how to compute a NIC network that represents (as best as possible) the original structure of the network in a localized manner. The idea is to start with the original network and either "add" or "remove" a small number of links such that the resulting network is a NIC network. By adding links we mean considering a new node as a neighbour, only for recolouring purposes, and by removing links we mean disregarding a given neighbour, also for recolouring purposes only.

By adding links, one could construct a NIC network or even complete a (non-planar) superset of a triangulation, which is known to have at most a cubic number of recolourings (see Theorem 12). However, the addition of links may involve pairs of nodes that are multiple hops away from each other. A purely local computation may not be possible and we may need to resort to a far more communication intensive distributed computation. See, for instance, the problem of computing the Delaunay triangulation [57] of a set of nodes, that also suffers from non-locality as links

connecting distant nodes need to be found throughout the entire network.

With additions discarded, we are left with the choice of removing links (or disregarding them for geometric recolouring purposes). The goal is to remove the minimum number of links that makes the network satisfy the NIC conditions. This way the topology of the original network is preserved as best as possible. The next theorem, however, states that it is hard to find such optimal configuration, even if centralized computation is allowed. More specifically, we prove the hardness of a decidability version of this problem, which directly implies the hardness of its minimization version. For this, we reduce the 3-SAT problem to our problem, using the notation from Garey and Johnson's book [29] as follows.

Problem 1. ***Non-Isolated Convex (NIC)***

Instance: (G, k)*, where* $G = (N, L)$ *is a network with* $|N| = n$ *and* k *is a numeric constant.*

Question: Is there a set of links $L' \subset L$ *such that* $|L'| \leq k$ *and* $G' = (V, L \setminus L')$ *satisfies the NIC conditions?*

Problem 2. ***3-SAT*** *[20]*

Instance: (V, C)*, where* V *is a set of variables, such that each variable in* V *can take either the value TRUE or FALSE, and* C *a set of clauses, such that every clause* $c \in C$ *consists of the disjunction of 3 variables in* V*.*

Question: Is there an assignment for all variables in V *such that all clauses in* C *have at least one true literal?*

A literal *is either a variable or its negation. We refer to such an assignment* $\mathcal{V}$ *as a* true assignment*.*

Theorem 19. *The NIC problem (Problem 1) is NP-Complete.*

To make the proof of this theorem simpler, we first provide a set of definitions and lemmas. See Figures 2.10, 2.11, and 2.12 for *gizmos*, *variable gadgets*, and *clause gadgets*, respectively, used for our proofs. The purpose of gizmos is to simplify the variable and clause gadgets. Gizmos represent tree like structures where "too many" links need to be removed to make the graph satisfy the NIC conditions. Notice that some nodes of the clause and variable gadgets have incident links represented with dashed lines. We called these *connecting links*. The purpose of connecting links, as well as variable and clause gadgets, will shortly become evident.

Lemma 20. *Let (V, C) be an instance of the 3-SAT problem and V_i an instance of the variable gadget for $v_i \in V$. V_i satisfies the NIC conditions after removing $|C|$ links if and only if exactly all* TRUE links *or all* FALSE links *are removed.*

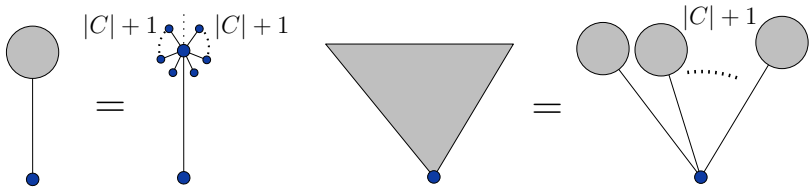

Figure 2.10: Gizmos used for the transformation from 3-SAT.

Proof. It is easy to verify that it is sufficient to remove either all TRUE links or all FALSE links for the gadget to satisfy the NIC conditions. Thus we concentrate on the necessity of removing one such set of links. Notice that the *centre node* is convex and has no convex neighbours. Removing the centre node's convex links serves no purpose because the gizmo's links then become its convex links, in which case at least another $|C| + 1$ links would need to be removed within the gizmos to make the network satisfy the NIC conditions. So, in order to keep the number of removed links

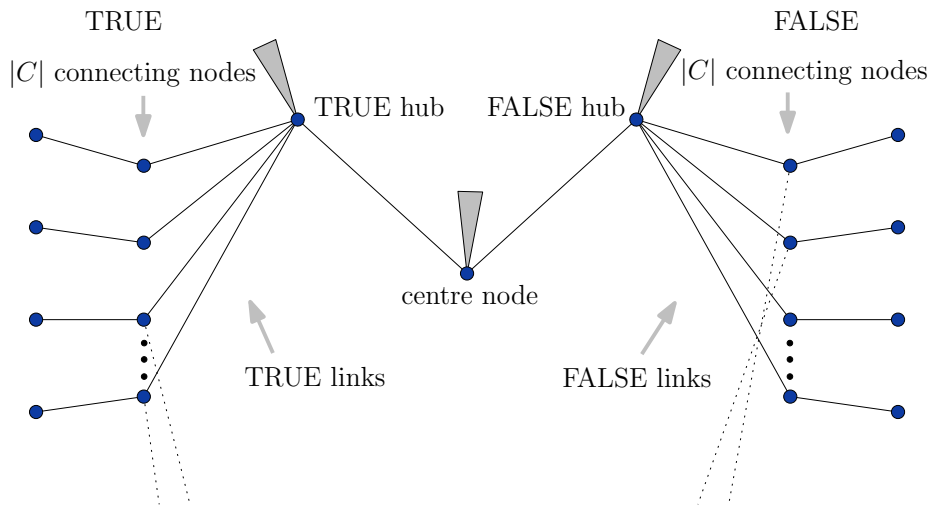

Figure 2.11: Variable gadget. The grey triangles are gizmos as defined in Figure 2.10.

smaller than or equal to $|C|$, we can only opt for making the centre node's neighbours convex, that is, either the TRUE or FALSE hub should become convex as well as the link that connects it with the centre node. This, in turn, can only be accomplished by removing all links incident to the hub node that are part of the gizmo or the TRUE or FALSE links, whichever corresponds. Once again, there would be too many gizmo's links to be removed so, there is no other choice but removing either the TRUE or FALSE links. □

Lemma 21. *Let (V, C) be an instance of the 3-SAT problem and C_j an instance of the clause gadget for $c_j \in C$. There exist exactly two links whose removal makes C_j satisfy the NIC conditions if and only if at least one $C_j[i]$ node, $i \in \{1, 2, 3\}$, is incident to a connecting link that is convex with respect to a node outside C_j.*

Proof. We first show that $C_j[i]$ being incident to a convex connecting link suffices for finding two links whose removal makes C_j satisfy the NIC conditions. Notice that all three $A_j[l]$, $l \in \{1, 2, 3\}$, are convex and have no convex neighbours. We show that by

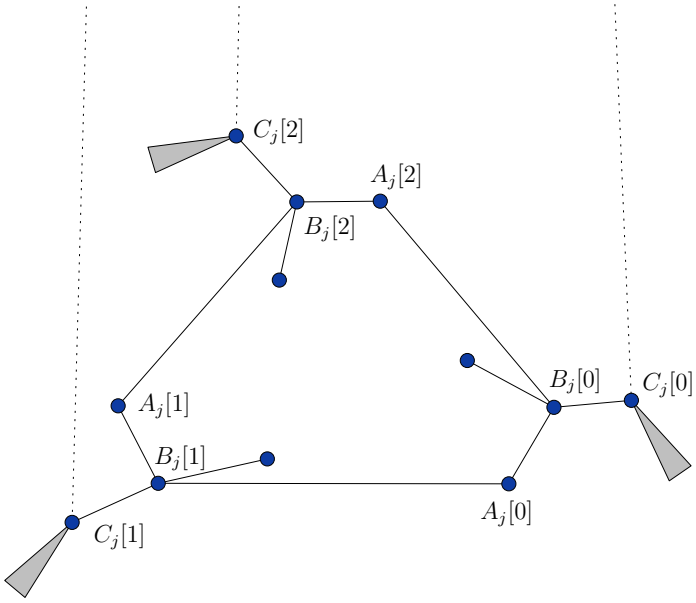

Figure 2.12: Clause gadget. The grey triangles are gizmos as defined in Figure 2.10.

removing two specific links the $A_j[l]$ can comply with the NIC conditions. Because $C_j[i]$ is incident to a convex connecting link, we can remove the link $\overline{C_j[i]B_j[i]}$. This creates a convex neighbour for both $A_j[i]$ and $A_j[i \oplus 2]$, with $\oplus$ defined as addition modulo three. After removing the link, $C_j[i]$ becomes convex; however, this does not create a conflict because it is adjacent through the convex connecting link to another convex node. In order to resolve for the remaining convex node, $A_j[i \oplus 1]$, the link $\overline{A_j[i \oplus 1]B_j[i \oplus 2]}$ is removed, leaving $A_j[i \oplus 1]$ with degree one. Notice further that there exists no single link whose removal resolves for all three $A_j[l]$, even if two or three connecting links are convex. Thus, the convexity of one connecting link is sufficient for the resolution of C_j by removing exactly two links.

It is also necessary that at least one connecting link is convex for making C_j satisfy the NIC conditions by removing only 2 links. This stems from the fact that no $\overline{C_j[i]B_j[i]}$ can be removed or else at least a second link must be removed to resolve for

$C_j[i]$, after it becomes convex without a convex neighbour. Consequently, two links would have been already removed and $A_j[i \oplus 1]$ would still not satisfy the required conditions. Because there is no other link that resolves for at least two of the $A_j[l]$, $l \in \{1, 2, 3\}$, making all three convex nodes satisfy the NIC conditions would require the removal of at least three links. This completes the necessity part of the proof. □

Proof. (Of Theorem 19)
Let (V, C) be an instance of the 3-SAT problem. Using the variable and clause gadgets we can construct a network G such that (V, C) can be answered in the positive if and only if the NIC problem (G, k) has a solution for a certain value of k to be defined below.

The construction of G is as follows. Initially G is the empty network. For every variable $v_i \in V$ we add a copy of the variable gadget V_i to G and for every clause $c_j \in C$ we add a copy of the clause gadget C_j. Then for every variable v_i and every clause c_j in which v_i appears we add a connecting link between a node $C_j[l]$, $l \in \{1, 2, 3\}$, and a connecting node in the TRUE side of V_i if v_i appears as a positive literal in C_j, or a connecting node in the FALSE side of V_i, otherwise. For the latter, we choose a node of V_i that still has no incident connecting links. The construction of the gadgets guarantees that there are enough nodes for this. In order to obtain the required angles at the connecting links, the embedding of G in the plane is such that connecting links are nearly vertical. This is achieved by placing all variable gadgets far up along the vertical direction and the clause gadgets far enough in the opposite direction. Since G does not have to be planar, we are not concerned about link crossings. G results in a network with $O(|V||C|^3)$ nodes and links, and spans a polynomial size area. Thus, G can be constructed in polynomial time.

Lemmas 20 and 21 establish that removing $|C|$ links from each one of the $|V|$ variables and 2 links from each one of the $|C|$ clauses produces a network that satisfies the NIC conditions if and only if all clause gadgets have at least one connecting link incident to a variable gadget node from which the corresponding TRUE link or FALSE link has been removed. Thus, by appropriately setting $k = |V||C| + 2|C|$ and setting the truth assignment for (V, C) as TRUE for each variable whose corresponding TRUE links have been removed and FALSE otherwise, one can state the following equivalent statement. (G, k) has a positive answer if and only if each clause gadget contains at least a negative literal connected to a variable gadget interpreted as FALSE, or a positive literal connected to a variable gadget interpreted as TRUE. That is, (G, k) has a positive answer if and only if (V, C) has a true assignment.

Thus, we have shown that the 3-SAT problem has a solution if and only if the associated instance of the NIC problem, which can be obtained in polynomial time, has a solution. This completes our proof. □

Because the optimal solution is hard to find, we study heuristic algorithms that eliminate a relatively small number of links. It is noteworthy that in the worst case even the number of link removals may be linear in the number of network links and quadratic in the number of nodes. An example network that exhibits this property is shown in Figure 2.13.

This example shows a network $N = (U \cup V, L)$ that defines a complete bipartite graph with subsets of nodes U (top) and V (bottom). Since both U and V have $O(n)$ nodes, L has $O(n^2)$ links. Let $N' = (U \cup V, L')$ be a NIC network obtained from N by link removals and let $u_i \in U, v_j \in V$ be a pair of nodes sharing a convex link. For $\overline{u_i v_j}$ to be a convex link, at least all $\overline{u_i v_k}$ links must be removed, with $v_k \in V$ and

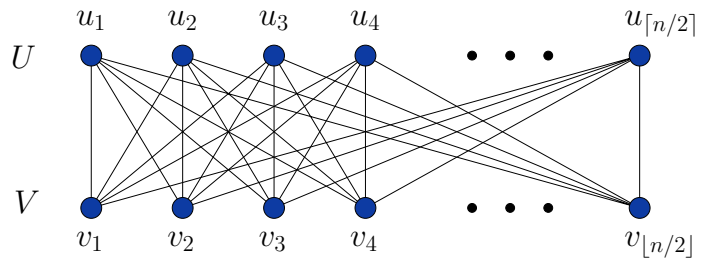

Figure 2.13: Worst-case example in which a quadratic number of links need to be removed in order to make the network satisfy the NIC conditions.

$1 \leq k < j$, or $j < k \leq \lfloor n/2 \rfloor$. The number of removed links is minimized when either $j = 1$, or $j = \lfloor n/2 \rfloor$, in which case, no link needs to be eliminated. Notice that in N' at most a second node $u_k \in U$ can share a convex link with v_j. Therefore, in at most four cases a node in U is not an isolated convex without eliminating any of its incident links. Similarly, one can argue that in only four cases a node in U is not convex isolated after eliminating one of its incident links in N. The same applies for the cases of eliminating two, three, and up to $\lceil n/2 \rceil/4$ links. Thus, the number of links that need to be eliminated for the nodes in U to satisfy the NIC conditions add up to quadratic as shown in the next equation.

$$|L| - |L'| \geq \sum_{j=0}^{\lfloor \lceil n/2 \rceil/4 \rfloor} 4j \geq 4(\lfloor n/8 \rfloor)(\lfloor n/8 \rfloor + 1)/2 = \Omega(n^2)$$

The previous example shows that optimal solutions, and heuristic solutions alike, cannot be assumed to eliminate a small number of links in the worst case. Therefore, we propose a simple heuristic algorithm that eliminates a small number of links according to our simulations. The heuristic algorithm for constructing the NIC network, the NIC Algorithm, is described next. The NIC Algorithm operates under the same assumptions as the geometric recolouring strategy: all nodes know their neighbours and the angle they define with respect to themselves. The algorithm consists of a

single protocol executed at all nodes.

Algorithm 2: NIC Algorithm

input : Network $G = (N, L)$ with bi-chromatic nodes.
output: Network $G' = (N, L')$ such that $L' \subseteq L$ and G' satisfies the NIC conditions.

NIC Protocol (executed at node p)

Step 1. Mark all links incident to p as *UNKNOWN*.

Step 2. If there is no message in p's message queue and there are still links marked as unknown, then send a message to each neighbour. The type of the message sent is either CONVEX or NON-CONVEX, depending on the convexity of the link with respect to p.

Step 3. If there is a message in the node's message queue, process the message according to its type:

CONVEX: the link through which the message was received is marked as *convex*. If the link was not marked as *convex* before, then a message is sent back to the sender indicating the convexity with respect to p.

NON-CONVEX: the link through which the message was received, l, is marked as *non-convex*. If l is convex with respect to p, and p is an isolated convex, then p removes l, sends a REMOVE message to the corresponding neighbour, and sends CONVEX messages over any other link that may have become convex after removing l.

REMOVE: the link through which the message was received, l, is removed.

Step 4. Go to Step 2.

Link marks, as used in the protocol, are relative to the node, that is, a link can be marked differently by each incident node. Notice that Step 3 of the protocol assures that no isolated convex node will remain connected to a non-convex node. The algorithm is guaranteed to terminate because links are always removed and never replaced. Note that a network empty of links satisfies the NIC conditions so, in the

worst case the algorithm terminate when all the links have been removed. Also, this protocol does not allow infinite loops because once a link is convex, it never becomes non-convex.

2.4.4 A hybrid strategy

We propose a simple strategy that combines the combinatorial and the geometric strategies and yields the best experimental results for certain degrees of connectivity, as will be seen in Section 2.4.5. The combination of the combinatorial and geometric recolouring strategies requires some extra care; otherwise the resulting strategy may not terminate, even though each separate strategy does.

For the *hybrid strategy* we define a SURROUNDED_HYBRID function that counts the number of neighbours of each colour. If the number of neighbours of the opposite colour is greater than the number of neighbours of its own colour, then it returns TRUE; if there are fewer neighbours of the opposite colour than its own colour, then it returns FALSE; otherwise (if there is a tie), the function returns the value of SURROUNDED_GEOMETRIC according to the geometric recolouring criterion. In other words, geometric recolouring is used to break the ties that may arise while performing combinatorial recolouring.

It is easy to see that this process yields a finite recolouring sequence if the geometric component considers a NIC network: the number of magenta links always decreases or remains the same (see the proof of Theorem 17). Also, while recolourings that preserve the number of magenta links occur, the colour-change number in the opposite chain cover decreases (see the proof of Theorem 10). It then follows that the number of recolourings is at most the multiplication of the maximum possible number

of recolourings for each method. This is formally stated in the following theorem.

Theorem 22. *Let $G = (N, L)$ be a network with bi-chromatic node set N. In the sequential model, the Recolouring Algorithm with parameter T and the* SURROUNDED_HYBRID *function terminates after $O(|N||L|^2)$ recolourings from the time of the last fault plus T.*

There may be more effective ways to combine the two recolouring approaches into a hybrid one; however, the previously described approach is the only one we know to guarantee termination. In the next section we present a "dishonest" method, called *cheating*, in which a centralized control over the recolouring process leads to a more effective way to combine the combinatorial and geometric approaches.

2.4.5 Experiments

Our experiments are based on the sequential model, that is, the probability that two nodes are executing an action simultaneously is zero. The experimental test bed consists of a set of connected networks generated at random with 100 nodes uniformly distributed over a 100×100 square grid. The links are defined as in a unit disk graph (UDG). We generate a set of 1000 random connected networks with unit distance Δ taking on values 15, 20, 25, and 30 times the width of a grid square. These distances have been chosen so that the network is k-connected with high probability for values of k ranging from 1 to 10: 15 corresponds to $k = 1$ and 30 to $k = 10$. The rest of the distances correspond to other k values between 1 and 10. The results plotted below are the average over each set of the 1000 randomly generated networks for each unit distance.

We first present the experimental results for the NIC algorithm. The mean ratio between number of links removed and total number of links is plotted in Figure 2.14 for different transmission radii (Δ). It is noticeable that the results improve as Δ increases, and therefore the number of links and the connectivity of the network also increase. Obviously, for higher values of Δ, the number of convex nodes tend to appear only at the boundary of the grid. According to the NIC Algorithm these are the only nodes from which incident links are removed. The graphs also include a set of bars around each value, representing the standard error of the mean. Because our set of samples is quite large (1000 networks), the errors are too small to notice in some cases.

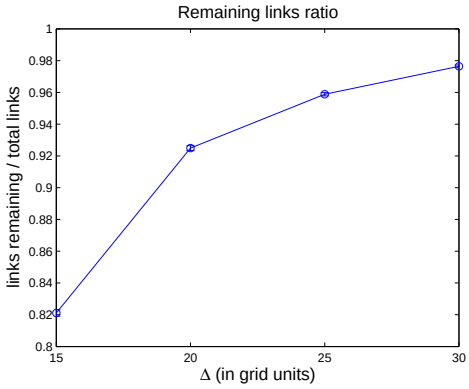

Figure 2.14: Average percentage of remaining links produced by the NIC Algorithm.

Next we present the results of the recolouring process for all the methods described above. For our experiments we have induced faults on the nodes that fall within a randomly chosen circular area within the grid. The circular area is defined by radii 10, 12, 14, 16, 18, and 20 times the width of a grid square, and is placed such that it falls at least half of its radius away from the border of the grid area. The latter is meant to eliminate the "border effect", that is, faulty nodes at the border are

more difficult to recover because of the smaller number and angle span of neighbours around them. The following graphs (see Figure 2.15) show the ratio of nodes that remain (or become) faulty after the recolouring process terminates over the different induced fault area radii. Notice that the results are represented on different scales for clarity. We also include the standard error of the mean on this graphs.

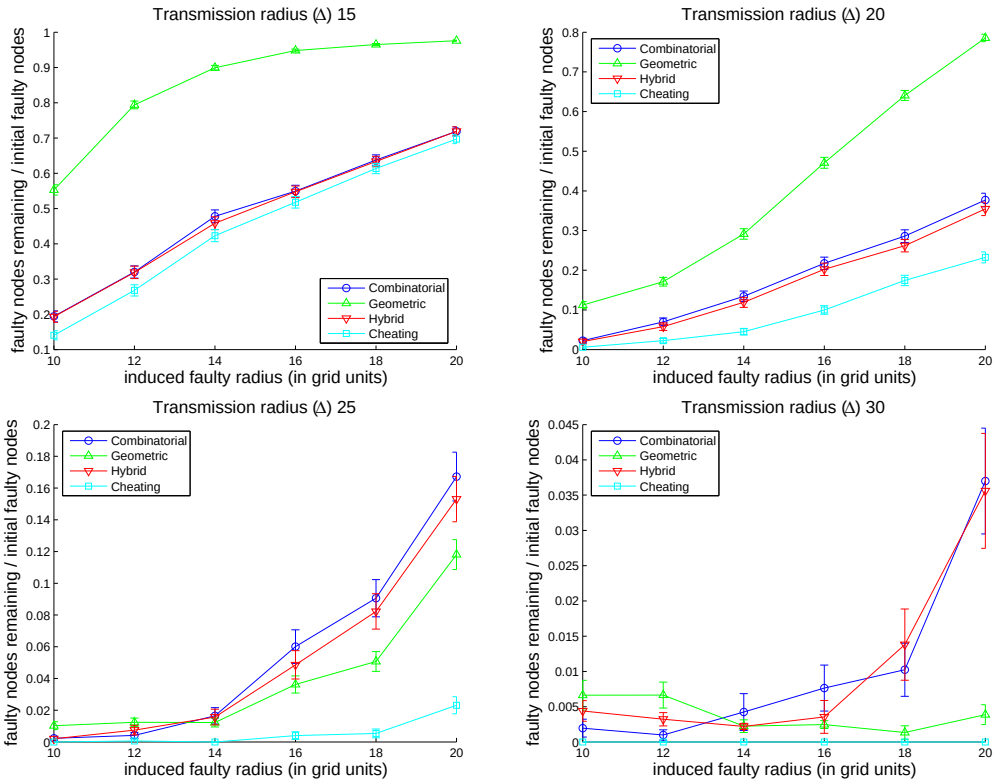

Figure 2.15: Average percentage of nodes that remain faulty after the application of different recolouring techniques.

In Figure 2.15 the method referred to as "Cheating" consists of the application of geometric recolouring, followed by combinatorial recolouring, then geometric recolouring one more time, and again combinatorial recolouring. Each run is performed after the previous one has stopped. In this sense, the cheating method is dishonest, because in a truly localized algorithm a node would have no information regarding

the global termination of recolouring, and therefore it cannot decide when to switch strategy. The cheating method has been shown with the sole purpose of exemplifying that there might be better ways to combine the combinatorial and the geometric methods than the hybrid approach shown here. However, how to do so in a localized manner while guaranteeing termination is unknown. Notice that the number of remaining faulty nodes after applying the "Cheating" becomes zero; thus, it partially disappears on the third graph and completely on the last graph as its value becomes zero.

Without considering the cheating method, the graphs show that the best results are in general given by the hybrid approach for small Δ values and by geometric recolouring for large Δ values. Therefore the best fault recovery strategy would use k, as in k-connectivity, for deciding when to apply the hybrid or the geometric recolouring method.

2.5 Open Problems

In the previous sections we have mentioned some open problems and conjectures. We summarize a list of these and other problems of interest.

- Find or improve bounds on the lengths of geometric recolouring of other types of networks. For example, Theorem 12 yields a $O(n^3)$ bound on the number of recolourings of dense non-plane graphs that contain a triangulation; however, no example matching this upper bound is known. Also, no tight bounds are known for geometric recolouring of networks such as UDGs or outer plane networks (i.e., networks in which all nodes are on the outer face).

- Discover the complexity of geometric recolouring when the threshold angle to consider a node as surrounded is greater than 180°. It has been conjectured that triangulations allow only recolouring sequences of linear complexity if the threshold considered is $\theta \geq 300°$.

- Characterize networks that have finite recolouring sequences through local properties at the nodes. These can be used for fault recovery in wireless networks. Maximum degree 3 networks and NIC networks are examples of such characterizations. Notice that other networks such as triangulations are not described locally at the nodes. It is not known however, if there exist other types of networks with similar properties that require smaller number of modifications (link removal or additions) than the NIC network and also guarantees finite or even linear bounds for recolouring sequences.

- Find other recolouring strategies suitable for fault recovery. Perhaps there are better hybrid approaches to combine combinatorial and geometric recolouring. The "cheating" method, as shown in our experiments, suggests that there may be more effective ways to do so.

2.6 Concluding Remarks

We have re-examined a geometric node recolouring method previously used for reclassifying points to obtain reasonable subdividing boundaries. We show tight bounds for geometric recolouring in networks that define triangulations, trees, and other types of graphs. The complexity of recolouring sequences on networks ranges from infinite for general (plane) networks to linear for trees.

We present an approach to fault recovery in wireless networks through recolouring. The more traditional combinatorial recolouring method is compared with the geometric recolouring and a hybrid approach that combines the two of them. We have shown that geometric recolouring requires a certain preprocessing (topology control) in order to guarantee termination. However, it is worth the effort as it provides, by itself or in its hybrid version, a more desirable outcome in terms of resulting faulty nodes. This has been shown through simulations on unit disk graph networks and a sequential model of recolouring suitable for asynchronous computation networks.

Chapter 3

Distributed Computation of Voronoi Diagrams

3.1 Introduction

Voronoi diagrams are among the important structures in computational geometry; in fact, they are "...second in importance only to the convex hull" according to O'Rourke [59]. The application areas of Voronoi diagrams are very diverse, including biology, chemistry, and geography, to name but a few. We study applications of Voronoi diagrams to wireless networks, and efficient distributed algorithms for their construction. In what follows we provide some definitions and properties that characterize Voronoi diagrams.

3.1.1 Voronoi Diagrams

Definition 18. ***(Voronoi region)*** *(adapted from [59]) Let* $N = \{p_1, p_2, \ldots, p_n\}$ *be a set of nodes in the plane. The* Voronoi region $\mathcal{V}(p_i)$ *of a node* $p_i, i \in \{1, \ldots, n\}$, *is the set of points that are closer to* p_i *than to any other node, i.e.,* $\mathcal{V}(p_i) = \{x : \|p_i - x\|_2 \leq \|p_j - x\|_2, j \neq i\}$.

In the previous definition, by a slight abuse of notation, we refer to a node or its planar coordinates by the same symbol. We will use this notation throughout; however, it will be clear from the context whether we refer to a node or its coordinates.

Note that according to the definition, the Voronoi regions are closed sets. Their boundaries define polygonal curves consisting of vertices and (possibly unbounded) edges. A direct consequence of the definition is that edges and vertices of Voronoi regions consist of line segments and points that are at equal distance from two or more nodes. (Points on the edges are equidistant to exactly two nodes, whereas points on the vertices are equidistant to three nodes, or more if nodes were not in a general position.)

Let N be a set of nodes. The union of edges and vertices of the Voronoi regions for all the nodes in N is the *Voronoi diagram* (VD) of N. We use the notation $VD(G)$ to refer to the VD of a network G. See Figure 3.1 (left) for an example Voronoi diagram. Notice that some edges have been clipped or entirely left out because of space limitations.

One interesting property of Voronoi diagrams is that its dual graph is the Delaunay triangulation (DT). This means that two Voronoi regions share an edge in the VD if and only if the corresponding nodes are neighbours in the DT. Figure 3.1 (right) shows a node set, its VD, and its DT (in dotted lines). Thus, computing the VD of

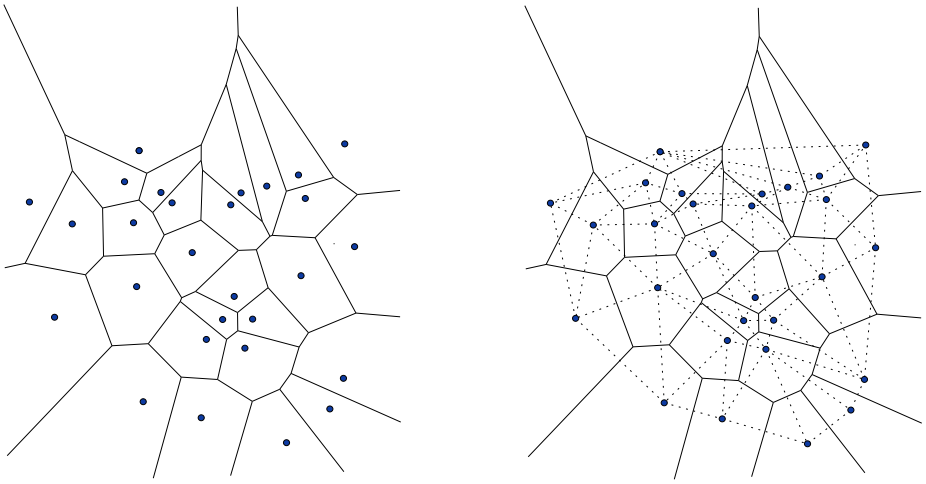

Figure 3.1: Voronoi diagram of a set of nodes (left) and the same diagram with its dual the Delaunay triangulation (right).

a network trivially yields its DT.

3.2 Literature Review

Voronoi diagrams have been excellently surveyed by Aurenhammer [6] and Okabe et al. [58]. Their reviews include algorithms, properties, and applications of VDs, as well as other related structures and extensions to various metrics and geometric spaces. Our review focusses on Voronoi diagrams in the context of wireless networks, a more recent topic not previously surveyed, to the best of our knowledge. In the following we present various applications of Voronoi diagrams to wireless networks and the distributed algorithms that have been considered for their computation.

3.2.1 Applications of Voronoi Diagrams to Wireless Networks

In addition to all the applications inherited from Delaunay triangulations, Voronoi diagrams have multiple applications to wireless networks of their own. For example, Sharifzadeh and Shahabi [76] propose a solution for querying WSNs using data aggregation mechanisms driven by VDs. Zhou, Das, and Gupta [88] present a scheduling mechanism that guarantees coverage of the network and is based on VDs. Chen, Hou, and Sha [15] address target tracking also using VDs. These authors have proposed algorithms for computing Voronoi diagrams of wireless networks. Their approaches are discussed next.

3.2.2 Background on Distributed Network Computation of Voronoi Diagrams

The computation of Voronoi diagrams by a centralized algorithm is a well-studied problem. There exist several efficient algorithms to this end (see for example Fortune's algorithm [26]). The situation is completely different –rather lame– when it comes to distributed computation of VDs. No efficient solution has been given for computing the VD of a connected network in a distributed manner. Unfortunately, this cannot be done for general networks by a localized algorithm because some edges of the VD may correspond to pairs of nodes that are too far apart.

We assume each node knows its exact location, which can be implemented through GPS or some other mechanism. Naturally, a trivial way to compute the VD is to flood the network with the information about the location of each node so that every node is able to compute the entire diagram independently. This, however, is the antipode of locality and leads to very inefficient algorithms. For example, a network with n

nodes would send $O(n)$ messages per node, for a total of $O(n^2)$ messages. Although this cannot be significantly improved in the worst case, we will see algorithms that improve this result for "realistic" networks.

In a distributed scenario, the goal is to get every node to know its Voronoi region. This is sufficient for many applications and seems to allow for much more efficient computation and communication than having all nodes know the entire VD. In this sense, there have been several attempts to efficiently compute the Voronoi diagram of a network in a distributed fashion.

Existing algorithms produce either approximations of the VD, compute VDs of special types of networks, or come without a correctness proof. For instance, Zhou, Das, and Gupta [88] propose a localized algorithm in which a node considers only the neighbours that are at most a constant k number of hops away from itself to produce an approximation of the VD for general connected networks (one in which the resulting regions may overlap). It is noteworthy that localized algorithms may produce the actual VD for a bounded region of the plane under assumptions that involve coverage of the entire network region. This is the case in the previously cited work, but it does not work in general. Bash and Desnoyers [7] also consider the VD of a bounded region; although their algorithm is not localized and does not require any assumptions regarding the coverage. Their algorithm, however, relies on the underlying routing scheme: this must be geographic routing [10] or more specifically its GPSR version [40]. The problem with this approach is that by computing the VD of a bounded region some important information may be missing: entire Voronoi edges could exist outside the region. Figure 3.2 (left) shows an example VD of a bounded region that does not contain all the edges of the complete VD, as well as

the corresponding missing edges on its dual Delaunay triangulation. (Notice that the network links have not been shown in the figure; it is assumed that the network is connected.) The fact that some Delaunay edges may be missing is unacceptable for network functions that rely on the complete VD or DT.

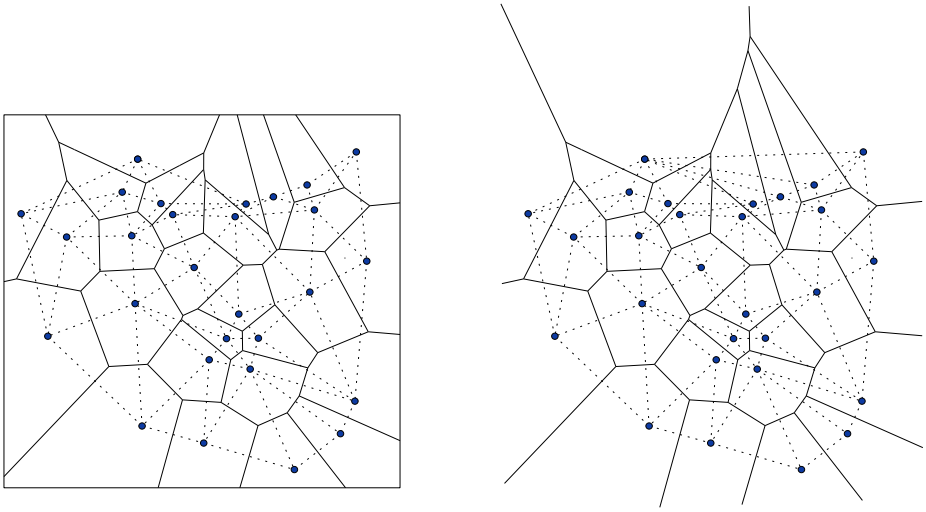

Figure 3.2: Voronoi diagram of a bounded region (left) and complete Voronoi diagram (right).

Another approach to constructing the Delaunay triangulation of a network has been proposed by Liebeherr, Nahas, and Si [46]. Their algorithm is based on a simple test for the local empty circle properties and collaboration among nodes. They did not, however, include a proof of correctness for their algorithm to show that in the end the correct DT is computed for any connected network. In previous work [57] we have proposed an algorithm that is somewhat similar to theirs. Both algorithms are similar in the sense that nodes mostly rely on cooperation for finding their Voronoi neighbours and computing their corresponding regions. Our algorithm, the Completely Cooperative (CC) algorithm has been proved to compute the actual VD whenever the underlying network is a UDG (or a slightly relaxed version of UDG).

We showed through simulations that the CC algorithm is more efficient in terms of communications than the algorithm proposed by Bash and Desnoyers.

3.3 A Worst-Case Optimal Algorithm

At this point a formal definition of the problem is in order. The input of our problem is the network $G = (N, L)$ with node set N and link set L. The desired output is the Voronoi diagram of N, in a way that each node knows its Voronoi region.

The following are a set of key graph theoretical definitions. Consider two nodes, $p, q \in N$, the *distance*, $dist(p, q)$, between p and q is the length of the shortest network path that connects p and q. The *diameter* of the network G, denoted as $diam(G)$, is defined as the maximum distance among all pairs of nodes, i.e., $\max_{p,q \in N} dist(p, q)$.

3.3.1 Lower Bounds

In order to establish the inherent complexity of the Voronoi diagram computation in a wireless network, we prove the following lower bounds on the time and communication costs. For the time lower bounds, we use the synchronous distributed network computational model as defined by Peleg [65].

It is worth mentioning that we do not consider compression of information for two reasons. First, none of the known algorithms for computing the $VD(G)$ in a distributed fashion make any use of compression. Second, the messages consist mostly of node coordinates, which can be real numbers and may exhibit a high entropy that do not allow significant compression.

Theorem 23. *Let $G = (N, L)$ be a wireless network with n nodes, such that G is the*

induced UDG of N and G is connected. Let also $D = diam(G)$. The communication cost for computing the Voronoi diagram of G is $\Omega(nD)$.

Proof. Consider an n-vertex "spider" network G constructed as in Figure 3.3 ($n = 32$ in this example). Define $d = (D-1)/2$. The spider graph consists of $\lfloor n/d \rfloor$ legs of length d plus one leg that has $n - d\lfloor n/d \rfloor$ length whenever n is not a multiple of d. Each leg is attached to the body at one of the joint nodes. The joint nodes, in the centre of the graph, form a cycle. We number the legs from 0 to $\lfloor n/d \rfloor$ in a circular order and the nodes within a leg from 1 to d, 1 being closest to the centre. Thus, we refer to a node on leg i with rank j within the leg as $p_{i,j}$. The Voronoi diagram of the spider network, which resembles a spider web, appears in dashed lines in the figure.

Notice that for all i, $p_{i,j}$ is a Voronoi neighbour of $p_{i+1,j}$ –addition on the leg number subscript is taken modulus $\lfloor n/d \rfloor + 1$ (or n/d, if n is a multiple of d). Let $\mathcal{A}$ be a distributed algorithm that computes the Voronoi Diagram of G. At some point during the computation $\mathcal{A}$ determines that $p_{0,d}$ and $p_{1,d}$ are Voronoi neighbours and computes the bisector between them. Since $\mathcal{A}$ is distributed, such computation is done as part of the protocol executed at a certain node p. Therefore, the location information of $p_{0,d}$ and $p_{1,d}$ need to arrive at p, which takes at least $dist(p, p_{0,d})$ and $dist(p, p_{1,d})$ messages respectively. It is easy to see that the function $dist$ defines a metric so, $dist(p, p_{0,d}) + dist(p, p_{1,d}) \geq dist(p_{0,d}, p_{1,d})$. From the definition of spider graph it follows that $dist(p_{0,d}, p_{1,d}) = 2d - 1$. Thus, $dist(p, p_{0,d}) + dist(p, p_{1,d}) > 2d-1$. Similarly, $dist(p, p_{0,d-1}) + dist(p, p_{1,d-1}) > 2d-3$ and, in general, $dist(p, p_{i,j}) + dist(p, p_{i+1,j}) > 2j - 1$. Adding all the messages together, we have

$$\sum_{i=0}^{\lfloor n/d \rfloor} \sum_{j=1}^{d} dist(p, p_{i,j}) + dist(p, p_{i+1,j}) > \sum_{i=0}^{\lfloor n/d \rfloor} \sum_{j=1}^{d} 2j - 1 = (\lfloor n/d \rfloor + 1)d^2 = \Omega(nd). \quad (3.1)$$

We conclude that there exist graphs for which the number of messages involved in solving the Voronoi diagram in a distributed manner takes $\Omega(nD)$ messages. This completes our proof. □

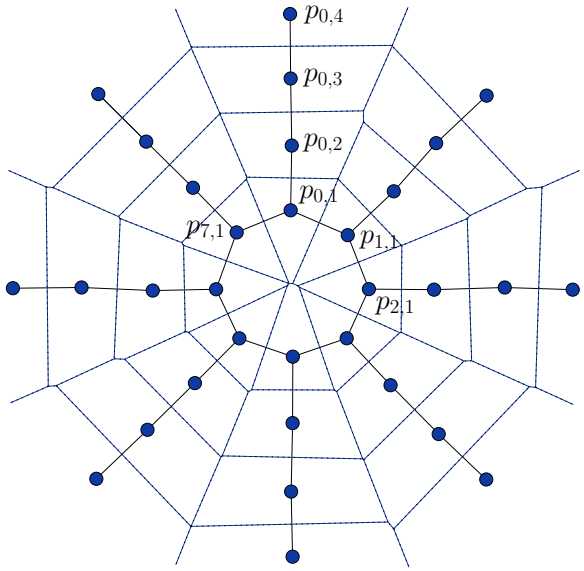

Figure 3.3: Spider network that serves as a lower bound example for the distributed computation of VDs.

Theorem 24. *Let G be a wireless network with n nodes such that G is the unit graph induced by $V(G)$ and G is connected. Let also $D = diam(G)$. The time required for computing the Voronoi diagram of G in the synchronous model is $\Omega(D)$.*

Proof. Obviously, the diameter of the network is a lower bound for the computation time if nodes that are $\Omega(D)$ hops away need to communicate with each other. The spider graph (see Figure 3.3) provides an example where the location of several nodes affect the Voronoi region of other nodes $\Omega(D)$ hops away. This argument can be formalized using the same tools as used for the proof of Theorem 23. □

3.3.2 The Leader-Based Algorithm

The Leader-based Algorithm, as will be seen in what follows (Algorithm 3), resembles a centralized computation of Voronoi diagrams. The leader is a distinguished node that controls the execution of a distributed algorithm and is known by all other nodes. Once the leader knows the location of all nodes, it can execute an efficient centralized algorithm for computing the Voronoi diagram, which could be Fortune's algorithm [26], for instance.

For now, we assume a leader is elected prior to the execution of our algorithm. There has been plenty of research on leader election in multiple scenarios. A brief discussion on leader election algorithms along with their complexities is given below.

The following terms are used in the description of Algorithm 3:

- ***queue***: Queue of messages of a certain node p that stores the messages received by p until they are processed.
- ***sendMsg(destination, information)***: Sends a message to node *destination* containing the data stored in $information$. The message is routed using a predetermined routing strategy and may involve several message transmissions (one per hop).

3.3.3 Analysis

It is obvious that Algorithm 3 is correct, provided that the input network G is connected. In the following, we show it is also worst-case optimal in terms of communication and time.

Algorithm 3: Leader-Based Algorithm

input : UDG network $G = (N, L)$ with a leader node.
output: Set of Voronoi regions for all nodes in N: each node learns its Voronoi region.

Leader Protocol

Broadcast a coordinates request to all network nodes
\\`Receive the information from all nodes`
while *not all coordinates received* **do**
  Read node coordinates from *queue*
Compute VD and store the Voronoi edges in *EdgeList*
foreach *edge* $\in$ *EdgeList* **do**
  `sendMsg`(Node on the left of *edge*, *edge*)
  `sendMsg`(Node on the right of *edge*, *edge*)

Regular Node Protocol

\\`Let` p `be an arbitrary node executing this protocol`
\\p `sends the requested information to the leader node`
Retrieve coordinate request message from *queue*
`sendMsg`(leader, p)
\\`Receive the description of its Voronoi region`
$\mathcal{V}(p) = \emptyset$ \\`empty list of edges`
while *edge information edge removed from queue* **do**
  Add *edge* to $\mathcal{V}(p)$
  if $\mathcal{V}(p)$ *contains the entire description of the cell* **then break**

Theorem 25. *Let $G = (N, L)$ be a wireless network with n nodes and let $D = diam(G)$. Algorithm 3 computes the Voronoi diagram of G by using $O(nD)$ messages and $O(n)$ time in the synchronous distributed network model of computation.*

Proof. We analyze separate steps of Algorithm 3.

1. **Broadcast request.** It is easy to see that broadcasting a message to the entire network takes $O(D)$ time and $O(n)$ messages. Receiving a one-message size reply from all nodes takes $O(n)$ time and $O(nD)$ messages, considering the worst case in which only one message is received per round.

2. **Local computation.** No communication is involved in the local computation of $VD(G)$. Also, it has been assumed that the time required for local computation is negligible compared to the communication delays. Thus, both time and communication complexities are considered null.

3. **Sending edge information.** The leader sends an overall linear size description (set of edges) of $VD(G)$. This takes $O(n)$ time and $O(nD)$ messages by sending one message per round such that the information is dispatched for one node at a time.

Adding all steps together, we obtain the claimed bounds. □

In the previous analysis, the messages are assumed to follow the shortest path to their destinations. Also, a synchronous model is considered. However, as each one of the steps involved can run asynchronously, the algorithm can be asynchronous if some care is taken to process messages in the appropriate order.

3.3.4 A Note on the Leader Election Problem

The leader election component of the algorithm could be incorporated as a first step; however, we prefer to consider it as preprocessing for two reasons: first, the leader may remain the same for multiple computations of the VD; second, as it turns out, computing the leader can be computationally more expensive than computing the VD itself. The most time efficient known algorithm for leader election in general networks is by Peleg [63], which uses $O(D)$ time and $O(|L|D)$ communication cost. Recently, a more efficient algorithm has been proposed for computing the leader of a network, with $O(D)$ and $O(|L| \min(D, \log n))$ time and communication complexities, respectively [41]; however, this algorithm is not deterministic and requires synchronization.

It would be convenient to have a leader election algorithm that uses only $O(nD)$ communication so that it does not dominate the complexity of the VD computation if used as a preprocessing step. Obviously, Peleg's algorithm leads to $O(n^2D)$ communication cost for dense networks. The algorithm by Gallager, Humblet, and Spira [28] offers a better communication cost, $|L| + n \log n$, for high diameter graphs, but takes $O(n \log n)$ time to compute. Their algorithm computes a spanning tree in order to find the leader. This is an expensive component in the algorithm, with $\Omega(|L|)$ communication cost.

The communication cost can be improved if the network is a UDG that has, for instance, quadratic number of edges and sublinear diameter. In this case, a spanner graph with $O(n)$ edges can be used as the underlying structure to execute Peleg's algorithm. Computing a spanner can be done with $O(n)$ communication cost, and seemingly $O(n)$ rounds, according to Alzoubi et al. [3]. In the overall this method yields $O(n)$ time and $O(nD)$ communication costs.

Note that in the previous argument, we refer to a spanner as a sparse graph that approximates the number of hops between two nodes within a constant factor, as opposed to the Euclidean distance or the path length. The reason for using this spanning metric is to abide to the unit message cost specifications of the distributed network model of computation. In this case, the diameter of the spanner is asymptotically equivalent to the diameter of the network.

Alternatively one can use the algorithm by Cidon and Mokryn [17]. Their algorithm takes $O(n)$ time and $O(n \log n)$ messages to elect a leader. In this case, messages are broadcasted to all neighbours, economizing in terms of communication. Considering their model, in which broadcasts have unit cost, their algorithm is more efficient than the algorithm by Gallager, Humblet, and Spira. This model is obviously suitable for wireless networks that communicate through radio signals.

There are also plenty of algorithms for leader election in highly dynamic networks, such as MANETs (see [11, 61, 84] for examples); however, no asymptotic complexity analysis has been provided and there does not seem to be a convention for a formal comparison of these methods. We also prefer not to use such methods for the leader election step because computing Voronoi diagrams cannot be done in a localized manner and our algorithm is more suitable for static networks, or networks that change slowly.

3.4 An Experimentally Efficient Algorithm

The algorithm presented in the previous section, although optimal in the worst case, does not take advantage of the network topology. That is, even if all pairs of adjacent Voronoi regions correspond to pairs of nodes that are also adjacent in the network,

the leader-based algorithm uses $O(nD)$ communication and $O(n)$ time. We propose a different algorithm that is more communication efficient for realistic networks.

The Completely Cooperative (CC) algorithm, as presented in [57] computes the Voronoi diagram of a network that is a connected unit disk graph (UDG). This algorithm, however, does not compute the Voronoi diagram of a general connected network. In the following we propose a more powerful version of the CC algorithm, the completely cooperative for connected graphs (CCCG) algorithm, that correctly computes the VD of any connected network.

3.4.1 The CCCG algorithm

We take advantage of the localization of the Voronoi diagram problem, that is, Voronoi neighbours are likely to be near each other if the network is relatively dense and the nodes are approximately uniformly distributed in a certain region. Thus, we avoid computing global features such as the leader as well as flooding the network with queries or information broadcasts.

Let N be a set of n nodes embedded in the plane. In the following we refer to a subgraph of the induced UDG of N as a general connected network G. Notice that one can take as unit distance the length of the longest edge of G and G will be guaranteed to contain all edges of $UDG(N)$. Then the edges of $UDG(N)$ that are missing in G can be removed. This is a convenient way to model a radio network in which nodes have identical transmission radius and obstacles block some of the communication links.

For clarity, we first describe the CCCG Algorithm (see Algorithm 4) in its simplest form. Some optimizations are presented below. The CCCG Algorithm consists of a

single protocol named *Compute_Region* executed at every node. The *Compute_Region* protocol defines the actions taken by any node p as it learns about other node q in the network. It also provides a mechanism for the refinement of successive approximations of the Voronoi region of p, until the correct region, $\mathcal{V}(p)$, is obtained. Let $\mathcal{V}^i(p)$, $0 \leq i \leq n$, be the i-th approximation of $\mathcal{V}(p)$ obtained during the computation. The initial approximation of the Voronoi region consists of the entire plane, that is, $\mathcal{V}^0(p) = \mathbb{R}^2$. Let H_{pq} be the halfplane that contains all the points that are closer to p than to a certain node q, more formally $H_{pq} = \{x \in \mathbb{R}^2, \|p - x\|_2 \leq \|q - x\|_2\}$, and $B_{pq} = \{x \in \mathbb{R}^2, \|p - x\|_2 = \|q - x\|_2\}$ the boundary line of H_{pq}. When p receives a message containing information from node q, p checks whether $\mathcal{V}^i(p)$, the current approximation of $\mathcal{V}(p)$, is not contained in H_{pq}. If so, $\mathcal{V}^i(p)$ is *refined* to obtain $\mathcal{V}^{i+1}(p) = \mathcal{V}^i(p) \cap H_{pq}$. See Figure 3.4 for an illustration of the refinement of $\mathcal{V}^i(p)$, to produce $\mathcal{V}^{i+1}(p)$ (shaded region). What happens next, defines the way information is disseminated across the network so that nodes learn about all their Voronoi neighbours.

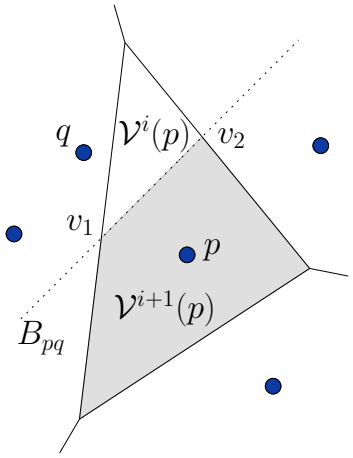

Figure 3.4: Refinement of the Voronoi region of node p with respect to node q.

When $\mathcal{V}^{i+1}(p)$ is computed, one or two new (possibly Voronoi) vertices appear. Let v_1 be a new vertex in $\mathcal{V}^{i+1}(p)$. Clearly, v_1 is the intersection point of two bisectors, B_{pq}, B_{pr}, for a third node r. Because v_1 could be an actual Voronoi vertex of $\mathcal{V}(p)$, meaning that v_1 is shared by $\mathcal{V}(p)$, $\mathcal{V}(q)$, and $\mathcal{V}(r)$, p sends messages to q and r about each other. This way, q and r learn about each other through p, if they did not previously know about the other.

In our previous work [57] we proved that the mechanism just described suffices for the algorithm to compute the correct VD of $UDG(N)$. However, additional communication between nodes is needed for the algorithm to work for a general connected network $G \subset UDG(N)$. The following modifications are introduced for this purpose. Node p keeps a list $Known$ of all the nodes it has learnt about during the computation. Whenever p receives a new message about node q, even if H_{pq} contains $\mathcal{V}^i(p)$, p checks the distance between q and every $r \in Known$. If q and r are less than unit distance apart and do not "seem to be adjacent", p sends messages to q and r about each other. The "seem to be adjacent" expression encapsulates a heuristic mechanism to detect, with certain accuracy, whether two nearby nodes are connected or the link between them is blocked. For this, every message containing the information of a node q includes a bounded, possibly incomplete, list of adjacent nodes of q, $q.adjacent$. So, from $q.adjacent$ and $r.adjacent$, p can estimate whether q and r are adjacent. Notice that if q appears in $r.adjacent$ or r appears in $q.adjacent$, q and r are obviously adjacent, but nothing can be inferred otherwise. The reason to keep the list of adjacent nodes bounded is to keep the message size bounded by a constant. If nodes do not seem to be adjacent through this mechanism, then unnecessary messages may be sent. This may be wasteful, but certainly does not introduce any error.

For simplicity, we define $p.info$ as a structure that contains all relevant information from p, that is, it includes p's coordinates and $p.adjacent$.

Next, a pseudocode description of the CCCG algorithm is provided to fill in the gaps of the previous explanation. The definitions of *queue* and the function $sendMsg$ are the same as for the Leader-based Algorithm (see Section 3.3.2)

Algorithm 4: Completely Cooperative for Connected Graphs (CCCG) Algorithm

input : Network $G = (N, L)$.
output: Set of Voronoi regions for all nodes in N: each node computes its own Voronoi region.

Compute_Region Protocol

```
1  begin
       \\p is the current node executing the protocol
2      Known = ∅ \\the empty list
3      V^0(p) = R^2
4      Broadcasts p.info to all adjacent nodes
5      while retrieve q.info from queue do
6          Refine V^i(p) with respect to node q to obtain V^{i+1}(p)
7          foreach vertex v ∈ V^{i+1}(p) and v ∉ V^i(p) do
8              Let r be the node such that v = B_pr ∩ B_pq
9              sendMsg( r, q.info )
10             sendMsg( q, r.info )
11         if q ∉ Known then
12             foreach r ∈ Known do
13                 if ||q − r||_2 ≤ 1 and ||p − q||_2 ≤ 1 and ||p − r||_2 ≤ 1 and
                   q ∉ r.adjacent and r ∉ q.adjacent then
14                     sendMsg( r, q.info )
15                     sendMsg( q, r.info )
16             Add q to Known
   end
```

3.4.2 Proof of Correctness

It is not hard to see that the CCCG Algorithm (Algorithm 4) terminates after a finite number of steps. Notice that every message sent is the result of the refinement of a Voronoi region, whose area decreases, or the result of a node that just discovered another previously unknown node and notified others about its discovery. So, the computation ends as the computed Voronoi regions converge to the correct regions and nodes learn about other nodes in the network. Next we establish the correctness of the algorithm.

Theorem 26. *Let $G = (N, L)$ be a connected network with $|N| = n$ nodes. The CCCG algorithm computes the correct Voronoi cell of every node in G.*

Before proving the theorem, we present some basic results. First we state a lemma that must have been proved somewhere else. We, however, include a simple proof herein for the sake of completion.

Lemma 27. *Let $G = (N, L)$ be a connected network. There exists a node $p \in N$ such that the subgraph of G induced by $N \setminus \{p\}$ is also connected.*

Proof. Consider T, a spanning tree of G. It is well known that T contains at least two leaves. Let p be a leaf of T. Obviously, removing p and its incident links does not disconnect G, proving the existence of the sought node. □

Lemma 28. *Let N be a set of nodes embedded in the plane, $p \in N$, and x a point in the plane. If $x \notin \mathcal{V}(p)$ then there exists a Voronoi neighbour of p, p_1, such that $\|p_1 - x\|_2 < \|p - x\|_2$.*

Proof. Consider the subset of nodes N' consisting of p and its Voronoi neighbours. Obviously the Voronoi region of p is the same in the VD of N and the VD of N'. Thus,

the Voronoi region of p does not contain x in either diagram. Naturally, in the VD of N', x falls in the Voronoi region of a node p_1. It follows that $\|p_1 - x\|_2 < \|p - x\|_2$, which proves the lemma. □

Proof. (of Theorem 26)

The proof is by induction on the number of nodes. The base case is for $n = 2$ since for one node there is nothing to compute. In this case, the two nodes share a link, by the hypothesis of the theorem. Therefore, they learn they are Voronoi neighbours of each other once the nodes broadcast their positions (line 4 of the CCCG Algorithm).

Assume $n > 2$. By Lemma 27, there exists a node p such that the subgraph of G, G^-, induced by $N \setminus \{p\}$ is connected. The CCCG Algorithm correctly computes $VD(G^-)$ by the inductive hypothesis. We show that the CCCG Algorithm finds a way to construct $VD(G)$ based on the fact that $VD(G^-)$ can be constructed correctly.

We first show that if p learns about one of its Voronoi neighbours, then it eventually learns about the other neighbours. Let p_1 be a Voronoi neighbour of p discovered through the CCCG Algorithm, and let $B(p, p_1) \subset B_{pp_1}$ be the bisecting edge that borders $\mathcal{V}(p)$ and $\mathcal{V}(p_1)$. Notice that $B(p, p_1)$ is either a semi-line or a line segment, because $n > 2$ (see Figure 3.5). Thus, $B(p, p_1)$ has an endpoint that is the Voronoi vertex v shared by p, p_1, and p_2 (i.e. $\|v - p\|_2 = \|v - p_1\|_2 = \|v - p_2\|_2$). Notice that the analysis is symmetric for both endpoints of $B(p, p_1)$, if it is a line segment. According to the algorithm, p learns about p_2 through p_1. Similarly, p learns about all its Voronoi neighbours in a contiguous order around p. Because the boundary of $\mathcal{V}(p)$ is a connected set, all the edges are eventually considered.

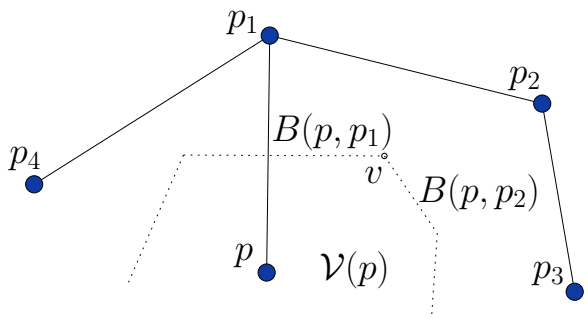

Figure 3.5: Elements of the proof of correctness of the CCCG Algorithm: discovering the entire Voronoi region description.

In order to complete the proof, we show that there exists a certain Voronoi neighbour that p necessarily learns about first through the application of the CCCG Algorithm. This holds even if p is not connected to any of its Voronoi neighbours by a link. Since G is connected, there exists a link $\overline{pq}$ that crosses the boundary of $\mathcal{V}(p)$. Let p_1 be a Voronoi neighbour of p such that $B(p, p_1)$ is the Voronoi edge crossed by $\overline{pq}$. If $G = UDG(N)$ then the link $\overline{p_1 p}$ is also in L given that $\|p - p_1\|_2 < \|p - x\|_2 + \|x - p_1\|_2 < \|p - q\|_2 \leq 1$, where x is the intersection point between $\overline{pq}$ and $B(p, p_1)$. Under the more general assumption that $G \subset UDG(N)$, the link $\overline{p_1 p}$ may not exist (see Figure 3.6, where some links are blocked by obstacles represented by shaded areas). Next we show that, in this case, p can discover p_1 through q.

Consider the circle $C(p, q)$ with diameter $\overline{pq}$. It follows that p_1 is in the interior of $C(p, q)$, given that p_1 lies on the circle centred at x that goes through p, $C(x, p, p_1)$, and that $C(x, p, p_1)$ completely falls in the interior of $C(p, q)$. The latter argument is a consequence of the centre of $C(p, q)$ being on the same line as x and p, but farther away from p. Thus, $\|p_1 - q\|_2 < \|p - q\|_2 \leq 1$. At this point we consider three possible cases and show that in each case, q informs p and p_1 about each other according to

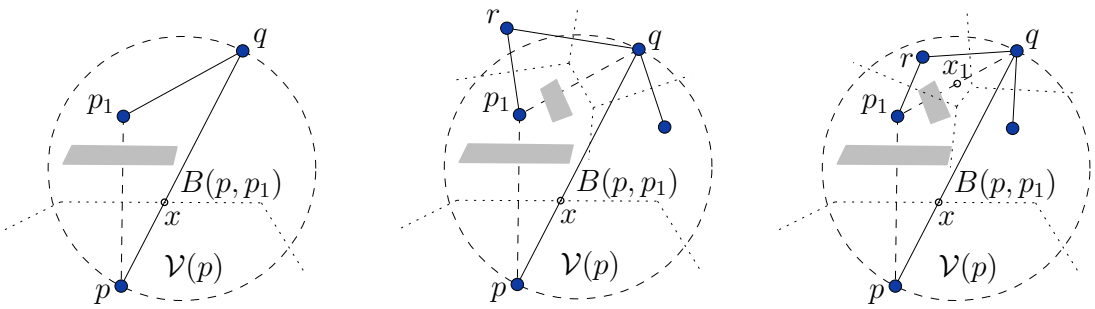

Figure 3.6: Elements of the proof of correctness of the CCCG Algorithm: discovering a Voronoi neighbour.

the CCCG Algorithm (see Figure 3.6).

Case 1, $\overline{p_1 q} \in L$. It is immediate that q learns about both p and p_1, and detects that they are not linked despite being less than unit distance apart. Thus, q informs p and p_1 about each other once it gets to know the second one of them (see lines 11 to 16 of the algorithm).

Case 2, $\overline{p_1 q} \notin L$ and q and p_1 are Voronoi neighbours. This is similar to the previous case, because q learns about p_1, given that $q, p_1 \in G^-$ and $VD(G^-)$ is computed correctly, by the induction hypothesis.

Case 3, $\overline{p_1 q} \notin L$ and q and p_1 are not Voronoi neighbours. In this case, the midpoint between q and p_1, x_1, must lie in the Voronoi region of another node. Thus, by Lemma 28, there exists a node r that is a Voronoi neighbour of q and is closer to x_1 than q. The CCCG Algorithm then guarantees that q learns about r and that r informs p_1 and q about each other, provided that r discovers p_1. Therefore, the problem reduces to whether r finds p_1. Notice that $\|r - p_1\|_2 < \|q - p_1\|_2 < \|p - q\|_2 < 1$. Similarly, one can prove that if r and p_1 are not

Voronoi neighbours, there exists another node s that is a Voronoi neighbour of r and is closer to the midpoint between r and p_1. This process repeats a finite number of times as the distance between the newly considered nodes and p_1 decreases monotonically. So, eventually a node that knows p and is a Voronoi neighbour of p_1 in G^- is found. This way, it is guaranteed that p gets to know a first Voronoi neighbour p_1.

So far, the inductive proof has been formulated as if the computation of $VD(G^-)$ occurred first, totally oblivious of node p, and then p started sending out messages and computing its own Voronoi region. Clearly, it is unlikely that messages follow such a contrived order. However, the message order determines only the number of refinements a Voronoi region is subject to, and therefore the number of false Voronoi vertices detected. For the computation to be correct it only matters that the notifications corresponding to the discovery of actual Voronoi vertices occur, no matter when. Furthermore, the notification that occurs whenever a node p discovers that the link between two other nodes, q and r, may be missing only needs to occur when q or r is a Voronoi neighbour of p. Thus, these notifications need not occur as a consequence of the discovery of false Voronoi neighbours. We conclude that the message order does not affect the correctness of the algorithm as all pairs of Voronoi neighbours discover each other.

This completes our proof. □

3.4.3 Optimizations and Other Implementation Details

As noted above, the CCCG algorithm is described in its simplest form. Some optimizations can be considered to make it more efficient. A simple heuristic consists of

processing closer nodes first.

Other optimizations are slightly more intricate. For example, if a node p, that is about to inform two nodes q and r about each other, finds q (respectively r) in the bounded list of adjacent nodes of r (respectively q), no notification is sent. This remarkably reduces the number of messages. Another improvement on the performance is given by not sending two simultaneous messages to possible neighbours q and r while trying to inform them about each other. Instead, a message is first sent to the closest one, say q without loss of generality, and then it is q who informs r, if required. This also reduces the number of sent messages because q and r may already know about each other through other node's notifications by the time q receives the message from p. Consequently, there is no need to inform r.

The CCCG Algorithm can be executed on a purely asynchronous network. However, a synchronous implementation allows to determine when the computation has been completed. For example, one can take the time bound derived in the next section and determine that the computation has ended because enough time has elapsed since the beginning of it.

3.4.4 Worst case analysis

Algorithm 4 does not offer the best performance in the worst case. We provide a worst case analysis to establish upper bounds on its communication cost and time complexities.

Since the CCCG algorithm does not rely on any specific routing scheme, we assume that routing occurs optimally –as in a spanner graph. That is, a message sent from a source node to a destination node travels along $O(D)$ links, where $D = diam(G)$.

We first analyze the complexity of the simpler version of the algorithm for UDG networks, that is, assuming all pairs of nodes that are less than unit distance apart share a link. In this case, messages need to be sent only as a consequence of successful refinements of Voronoi regions. Recall that refinements are based on the intersection of bisectors between pairs of nodes. Thus, the communication cost of computing the VD is $O(ID)$, where I is the overall number of bisector intersections found throughout the execution of the algorithm.

A first bound for I is given by the complexity of the arrangement of bisectors between all pairs of nodes. Shamos and Hoey [75] provide an $O(n^3)$ bound on such arrangements by bounding the number of Voronoi regions in all (higher) order Voronoi diagrams. It follows that $I \in O(n^3)$. A tighter bound can be obtained by noticing that in Algorithm 4 not every pair of bisectors is considered for intersection. In fact, a node n only considers bisectors of other nodes with itself. In the worst case, n finds new intersection points with all such bisectors. Therefore, $I \in O(n^2)$.

We now consider the cost of messages used to detect missing links on the UDG. Assume, for simplicity, that the bounded list of neighbours that comes along with each message is long enough to provide a good estimate of whether two nodes are adjacent or not. Let B be the number of blocked or missing links. At worst, each of the $n-2$ other sensors sends a message to the two nodes affected by a broken link to notify the problem. Each message travels $O(D)$ links, which lead to an overall $O(nBD)$ messages. Obviously, a large number of missing links affects the communication cost remarkably. The overall communication bound is stated in the following theorem.

Theorem 29. *Let $G = (N, L)$ be a wireless network with n nodes such that G is a subgraph of the induced unit disk graph of N and G is connected. Let B be the number*

of edges missing from G with respect to the UDG of N, and $D = diam(G)$. Algorithm 4 computes the Voronoi diagram of G with $O(n^2D + nBD)$ communication cost.

Note that this bound is worst than the bound provided for the Leader-based Algorithm and even the naive algorithm that floods the network with information about all nodes. This, however, is a worst case bound. In the next section we show that the actual communication cost is much lower –approximately linear in the number of nodes– for realistic networks.

The following example illustrates a situation in which the $O(n^2D)$ communication bound is attained for UDGs. See the five-node example provided in Figure 3.7. The example can be generalized to n nodes such that p_n is a Voronoi neighbour of all other nodes. In the same way, node p_{n-1} would be a neighbour of all other $n-2$ nodes to its left, had p_n not existed. Consider then a straightforward tracing of the CCCG Algorithm on such example and the following message processing order. Assume that all messages in or out of p_i, $2 \leq i \leq n$, are not processed until the VD of the nodes $\{p_1, \ldots, p_{i-1}\}$ on its left has been completed. Then messages from p_i are processed. As p_i is a neighbour of all $\{p_1, \ldots, p_{i-1}\}$, $\Omega(i^2)$ communication cost is involved in sending the information of all these nodes to p_i. Summing out the cost for every node in this order adds up to $\Omega(n^3)$, or $\Omega(n^2D)$, equivalently. The proposed message processing order simplifies the analysis of this example; however, any order will produce the same asymptotical bound.

A generalization of Figure 3.7 also provides a worst case example of the CCCG Algorithm for general connected networks. Assume all pairs of nodes are less than unit distance apart and that the only links that are not blocked are those appearing as dotted lines in the figure. In this example every node learns about every other

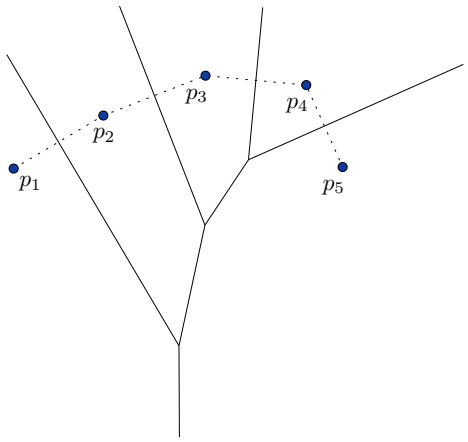

Figure 3.7: Worst case example for the CCCG Algorithm applied to a UDG network.

node. Thus, a rough analysis shows that every node detects every missing link and proceeds to inform every pair of nodes by sending them a message that may involve at most D transmissions. In the overall $\Omega(nBD)$ transmissions are involved, matching the provided upper bound.

A realistic worst-case upper bound on the time complexity of Algorithm 4 would involve the queueing delays that messages are subject to whenever they need to travel multiple hops to reach their destination. The very pessimistic assumption that each transmission takes a round, as in a synchronous model, leads to a trivial time upper bound $O(n^2D + nBD)$ for the CCCG Algorithm. We leave the derivation of tighter time bounds for future work. In the present work, we concentrate on the communication cost for being the most critical aspect of distributed computation in wireless networks.

3.4.5 Experimental Results

In previous work [57] we have shown that a simpler version of the CCCG algorithm for UDG networks outperformed, in terms of communication, that of Bash and Desnoyers [7]. In what follows we refer to the algorithm of Bash and Desnoyers as the BD07 Algorithm. To be consistent we compare the performance of the CCCG and BD07 algorithms on a set of connected networks. We also show how the communication cost changes with the size of the network. This is used to compare the CCCG Algorithm and the Leader-based Algorithm.

For the first experiment, we measure the average number of transmissions produced by the CCCG and the BD07 algorithms over a set of 1000 random networks. The networks consist of 100 nodes placed at random (uniformly) on a 100×100 square grid. The links are determined, as in the UDG, by a distance equal to 25 grid units. We include a set of obstacles in the shape of bars of length equal to 5 grid units, placed at random to block some of the communication links. This makes the scenario more realistic as such networks are more general than UDGs. (A similar setting has been used before in [57, 7].) With these settings, the average node degree is approximately 14. In order to make a fair comparison of these algorithms, we have used geographic routing for both, more specifically, the GPSR version of geographic routing (see [40]). The following graph (see Figure 3.8) summarizes the results over sets of obstacles of different sizes (ranging from 10 to 100). Notice that the CCCG performs better than BD07 when the number of obstacles is up to approximately 90 percent the number of nodes. We have also plotted bars around each value, representing the standard error of the mean.

For the next experiment we also compute the mean communication cost over a

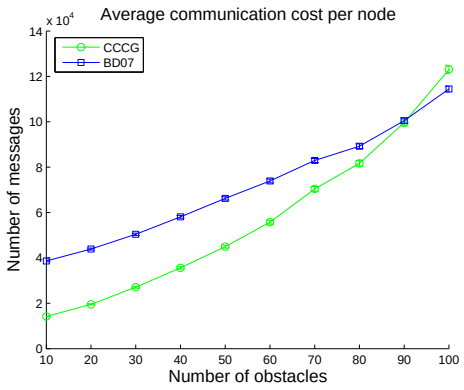

Figure 3.8: Results of the CCCG and BD07 algorithms on random networks for various obstacle set sizes.

set of 1000 random networks. In this case, however, the number of nodes ranges from 100 to 300 at increments of 25. The grid area is chosen as to maintain a constant density of nodes. More specifically, a set of n nodes is embedded in a $N \times N$ grid, where $N = \sqrt{n} * 10$. This leads, for instance, to a 100×100 size grid for 100 nodes as in the previous experiment. Figure 3.9 shows the resulting communication cost for two obstacle densities. The standard error of the mean, represented by bars, is relatively small for this sample size (1000 networks).

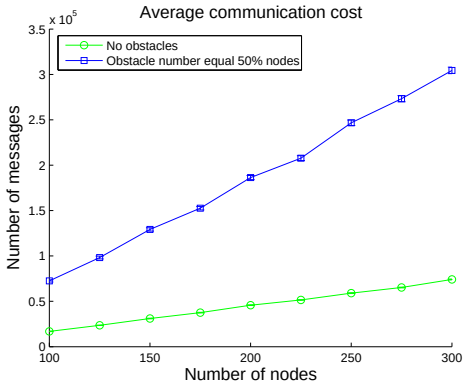

Figure 3.9: Results of the CCCG Algorithm for two obstacle densities and different network sizes.

It can be observed from the graphs (Figure 3.9) that the number of transmissions is nearly linear in the number of nodes. This suggest that the CCCG Algorithm is more efficient on random networks than the Leader-based Algorithm. Recall that the Leader-based Algorithm uses $O(nD)$ communication, where D is the graph theoretical diameter, or hops diameter.

In the following we derive bounds for D to highlight the differences between the complexity of both algorithms. The Euclidean diameter of the set of points embedded in the grid is at least $\Omega(\sqrt{n})$. This means that in the worst case the number of hops between two nodes is at least $\Omega(\sqrt{n})$, as the transmission radius is constant (25 grid units). (For more general random geometric graphs, similar bounds have been provided by Ellis, Jia, and Yan [24].) Thus, the communication cost used by the Leader-based Algorithm is $\Theta(n^{3/2})$, which is obviously super-linear. We may conclude that the Leader-based Algorithm has worse complexity than the seemingly linear experimental bound obtained for the CCCG algorithm.

3.5 Open Problems

The following open problems can be readily identified from the previous discussion:

- Design a time and communication efficient algorithm for the leader election problem in general networks.
- Derive formal average case bounds for the communication cost of the CCCG Algorithm.
- Derive tighter time complexity bounds for the CCCG Algorithm.

- Design a distributed network algorithm for computing Voronoi diagrams that is efficient in the average case as well as in the worst case.

3.6 Concluding Remarks

We have presented two simple algorithms for computing Voronoi diagrams in wireless networks. The Leader-based Algorithm offers asymptotically optimal communication cost in the worst case, as well as time efficiency in the synchronous model. The second algorithm, the CCCG Algorithm, has been designed for efficient communication in more realistic networks. Its communication cost has been estimated experimentally, showing a more desirable behaviour as compared to the Leader-based Algorithm on random networks. The CCCG Algorithm has also been compared to the algorithm proposed by Bash and Desnoyers [7], the BD07 Algorithm, which computes an approximation of the Voronoi diagram and has been tested in the same context as ours. Our algorithm (CCCG) is more efficient if the network is a UDG or a few links are missing from the UDG induced by the set of nodes. We conjecture that further optimizations of the CCCG Algorithm may lead to better efficiency even for arbitrary connected network. Two things to keep in mind when comparing the CCCG and BD07 algorithms is that ours computes more, as it finds all the edges of the Voronoi diagram, and may be trivially made more efficient by using more efficient routing algorithms than geographic routing.

Chapter 4

Hamiltonian Cycles in Solid Hexagonal Grids

4.1 Introduction

A *Hamiltonian circuit* or *Hamiltonian cycle* (HC) of a network is a simple closed path that contains all nodes. The problem of deciding whether a graph has a Hamiltonian circuit, the HC problem, is known to be NP-Complete (see Garey and Johnson's book [29] for a proof). In fact, this problem is hard even for special types of graph such as planar graphs and grid graphs. Nearly three decades ago, Itai, Papadimitriou, and Szwarcfiter [38] proved that the HC problem is hard for square grid graphs. The hardness of the HC problem for triangular grids was later proved by Polishchuk, Arkin, and Mitchell [69]. Recently, we proved that this problem is similarly hard for hexagonal grids [37]. The proof for the hexagonal case is slightly more involved, with a history of efforts devoted to proving the hardness of the HC problem for relaxed versions of hexagonal grids (see [13] for example).

Finding a HC in a solid square or triangular grid can be done in polynomial time. In the present work we address the HC problem for solid hexagonal grids. Although this problem remains open, our results bring us a step closer to its solution. One of the reasons no simple algorithm has been found to this end is that using the same tools as used for the square grid case seems to yield a more involved algorithm, as noted in Section 4.3.3.

All the previously cited results are in the context of centralized computational models, as opposed to distributed ones. It is not possible to solve the HC problem in a truly distributed manner, because the description of the problem itself involves global knowledge of the network: the circuit must visit all nodes. Thus, we study the HC problem from a centralized perspective.

This chapter continues with a review of the literature on the topics of Hamiltonian cycles and hexagonal grids as applied to several wireless network scenarios. Next we present our approach to the HC problem for solid hexagonal grids. Finally, we propose an algorithm for finding a particular type of subgraphs of a network, the so-called k-factors, with applications to the HC problem.

4.2 Literature Review

We separately review the applications of solid hexagonal grids and Hamiltonian circuits to wireless networks. Both these topics have been subject of research in the wireless networks context in recent years. We also review the history of the HC problem for other solid grids and introduce an important tool for the computation of HCs in graph, namely, the 2-factors of a graph.

4.2.1 Hexagonal Grid Graphs and Wireless Networks

Hexagonal grids have various applications to wireless networks communication as they appear in different such scenarios. For example Takahara, Xu, and Hassanein [80] proved that efficient network coverage can be achieved when nodes are placed approximately on grid points, for triangular, square, and hexagonal grids. Caragiannis et al. [13] have used configurations that are approximately hexagonal grids for embedding nodes with directional antennas. Such embeddings admit Hamiltonian circuits and therefore, strongly connected networks.

In other cases, the hexagonal grid is not defined by the nodes themselves, but by other points of interest. For example, hexagonal networks are widely used because of their constant node degrees and simple efficient routing algorithms [70, 56]. Not to be confused, the nodes of a hexagonal network are not placed on the hexagonal grid vertices but on the vertices of its dual triangular grid. However, the points where the transmission signal is weakest are the vertices of a (scaled) hexagonal grid. (These are also the vertices of the Voronoi diagram of the network.)

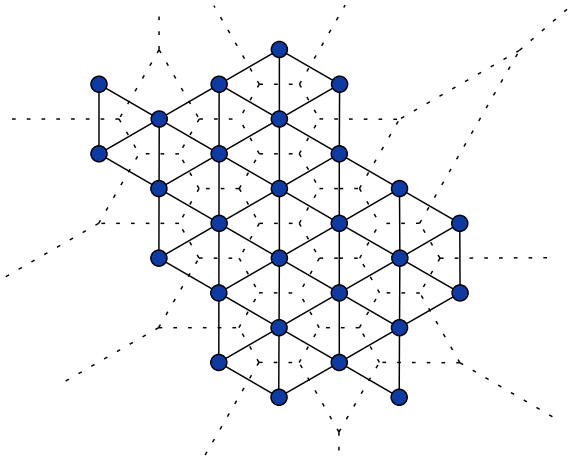

Figure 4.1: Triangular grid network and its Voronoi diagram (in dashed lines).

The idea of wireless networks being solid grids also seems very natural as wireless networks ideally have no "holes" or "blind spots". More formally, all points in the network area should be covered by at least one node. This motivates the study of solid hexagonal grids.

4.2.2 Hamiltonian Circuits and Wireless Networks

Hamiltonian circuits have also found various applications in wireless networks. One such example has been mentioned in Section 4.2.1, with the purpose of guaranteeing network connectivity. The following are other applications of HC to wireless sensor networks (WSNs) as suggested by several authors. Somasundara, Ramamoorthy, and Srivastava [79] propose a solution based on HCs for an entity that traverses the network collecting information at every node to avoid buffer overflows. Such entity could either be a physical device or a software agent consisting of a program to be executed at each node. In his thesis, Hill [35] recommends a solution to monitor network security based on HCs, as part of an architecture for WSNs. Patil, Das, and Nasipuri [62] present a mechanism for collectively detecting signals in WSNs that also uses HCs.

Because of the arguments presented in Section 4.2.1, it makes sense to study HCs in solid hexagonal grids as a special case of a favourable network topology. Another reason is that it has been conjectured that the HC problem for solid hexagonal grids can be solved in polynomial time (see Section 4.2.3). If this is the case, solid hexagonal grids would provide a very convenient topology for implementation of a wireless network.

4.2.3 Hamiltonian Circuits in Solid Grid Graphs

As previously mentioned, there are polynomial time algorithms for the HC problem in some cases of solid grid graphs. In this section we summarize those results. We begin with a definition that may be beyond basic graph theory.

Definition 19. ***(k-factor)*** *Let $G = (N, L)$ be a graph. A* k-factor *of G is a spanning subgraph of G in which every node has degree k.*

For instance, a 1-factor is a perfect matching and a 2-factor is a collection of cycles that spans the network. According to this definition, a Hamiltonian circuit can be defined as a 2-factor with a single connected component.

Finding HCs in solid triangular grids is relatively simple: they can be constructed systematically through a series of local transformations. In fact, the decidability version of the problem is trivial as all solid grid graphs have a HC excepting only one particular case, the *Star of David* [69]. For solid square grids the algorithm is more involved. Umans and Lenhart [83] propose a construction that starts with a 2-factor of the graph. Then they repeatedly apply a set of (global) reconfigurations to obtain new 2-factors with smaller number of connected components until a HC is found, whenever one exists.

It has been conjectured [4] that the HC problem on solid hexagonal grids can be solved in polynomial time. However, to date this remains an open problem. It seems that in the same way the NP-Completeness proof for the HC problem on solid grids is more involved for the hexagonal case, finding algorithms for the HC problem in solid hexagonal grids is more complicated.

In our attempt to solve the HC problem for solid hexagonal grids, we also consider the 2-factor approach. To this end we propose an algorithm to compute 2-factors that

generalizes to k-factors of graphs (see Section 4.4). In fact, we introduce the concept of *simple k-factor*, which simplifies the work, and the proofs for the HC problem.

4.3 Towards an Algorithm to Compute the Hamiltonian Cycle of Solid Hexagonal Grids

We present a collection of results that provide some insight on the HC problem for solid hexagonal grids. Some of our results are based on the works of Sheffield [77] and Umans and Lenhart [83] on the square grid case. We believe the complexity of the HC problem for solid hexagonal grids is similar to the complexity of solid square grids, perhaps slightly more complex as will be argued in Section 4.3.3.

In the following we present some necessary conditions for solid hexagonal grids to have Hamiltonian cycles. Next we introduce the preliminaries of transforming 2-factors for the sake of constructing HCs. Then we present an exploration of the approach used for the square grid case and its applicability to hexagonal grids, with some insight as to where the complexity of the two problems diverge.

4.3.1 Necessary Conditions

An algorithm for finding a Hamiltonian cycle could start by checking simple sufficient and necessary conditions in order to easily classify graphs into those that are and are not Hamiltonians. In their book, Hsu and Lin [36] review some well known sufficient conditions for a graph to be Hamiltonian: these include theorems by Dirac, Ore, and Chvátal and Erdös. They also provide necessary conditions. For example, for a network to be Hamiltonian it must be 2-connected. Also, if it is bipartite, it must

have the same number of nodes in each partition. The following necessary condition is stated more formally.

Theorem 30. *(Adapted from Hsu and Lin [36]) Let $G = (N, L)$ be a Hamiltonian network and S a subgraph of G. Then $G \setminus S$ has at most $|S|$ connected components.*

Unfortunately the sufficiency conditions mentioned above, as adopted from general graph theory, do not apply to solid hexagonal grids. On the other hand, the necessary conditions are either very hard to check, as it appears in the previous theorem, or are trivial –although the fact that the two partitions must have the same number of nodes may be of some help (see Figure 4.2 (left)). Figure 4.2 shows three example solid grid networks with different classifications according to the existence of 2-factors and HCs: (left) a network that has no 2-factors, (middle) a non-Hamiltonian network with a 2-factor, and (right) a network with a Hamiltonian cycle. In what follows we present some other necessary conditions that provide more insight on the hamiltonicity of hexagonal grids.

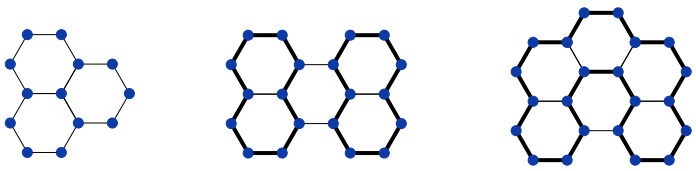

Figure 4.2: Example solid hexagonal grids with 2-factors (in thick lines).

Since we are adopting a 2-factor approach to the HC problem, the following condition is obvious: for a network G to be Hamiltonian it must have a 2-factor. It is not hard to check whether a network has a 2-factor; this can be done in polynomial time as shown in Section 4.4. Other necessary conditions are derived next.

A simple cycle C defined on a hexagonal grid separates the plane into two disjoint regions. We refer to the unbounded region as the *exterior* of C and the bounded

region as its *interior*. The *length* of the cycle, $|C|$, is defined as the number of nodes that form the cycle. The area of the interior of C, denoted by $\mathcal{A}(C)$, is the number of hexagons that define the bounded region.

Lemma 31. *Let $G = (N, L)$ be a solid hexagonal grid network, and let C be a subgraph of G that consists of a simple cycle. Let I be the set of nodes in N that fall in the interior of C. $|I|$ is even if and only if C consists of $4k + 2$ nodes, $k \in \mathbb{N}$.*

Proof. First notice that because G is bipartite $|C|$ is even and can be either written as $4k$ or $4k + 2$, $k \in \mathbb{N}$. We continue the proof by induction on $\mathcal{A}(C)$. The base case is for $\mathcal{A}(C) = 1$; thus, $|C| = 6$. The lemma trivially holds in this case as the number of nodes in the interior of C is 0.

By inductive hypothesis we assume the lemma holds for any C^- such that $\mathcal{A}(C^-) < m$. We prove it holds for any C with $\mathcal{A}(C) = m$. We use the intuitive idea of removing one hexagon from C to obtain at most two cycles C'_1, C'_2 with area smaller by at least one hexagon. The union of the interior node sets of C'_1 and C'_2 is denoted I'. To better define the removal operation we remove the leftmost-topmost hexagon, h, as shown in Figure 4.3. There are four different cases to consider:

Case 1: There are 2 nodes on the outer face that are only incident to h. $|I'| = |I| - 2$. The length of the cycle is preserved, that is, $|C'_1| = |C|$. By inductive hypothesis, the lemma holds for C'_1. Then, as the parity of I and I' is the same, the lemma also holds for C.

Case 2: There are 3 nodes on the outer face that are only incident to h. $|I'| = |I| - 1$ and $|C'_1| = |C| - 2$. By inductive hypothesis, the lemma holds for C'_1, that is, $|C'_1| = 4k + 2$ and $|I'|$ is even, or $|C'_1| = 4k$ and $|I'|$ is odd. In the former case

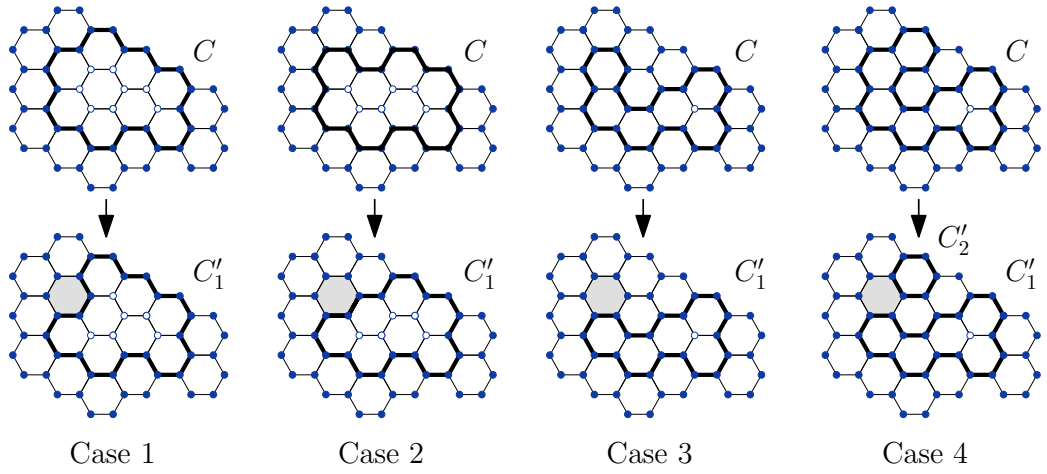

Figure 4.3: Proof of Lemma 31: three cases resulting from hexagon removal.

$|C| = 4(k+1)$ and $|I|$ is odd, and in the latter $|C| = 4k+2$ and $|I|$ is even, which proves the lemma.

Case 3: There are 4 nodes on the outer face that are only incident to h. $|I'| = |I|$ and $|C'_1| = |C| - 4$. It follows that the lemma holds for C, as the number of interior nodes is preserved as well as the divisibility by 4 of the length of the cycle.

Case 4: There are 2 nodes on the outer face that are only incident to h, and the removal of h creates two cycles. $|I'| = |I|$ and $|C| = |C'_1| + |C'_2| + 2$. By inductive hypothesis, the lemma holds for C'_1 and C'_2. Thus, if $|I'|$ is even, the number of points of I' in the interior of C'_1 and C'_2 are either both even or odd. If they are even, then $|C'_1| = 4k_1 + 2$ and $|C'_2| = 4k_2 + 2$, $k_1, k_2 \in \mathbb{N}$, if they are odd then $|C'_1| = 4k_1$ and $|C'_2| = 4k_2$. The former implies that $|C| = 4(k_1 + k_1 + 1) + 2$ and the latter implies that $|C| = 4(k_1 + k_1) + 2$. In either case, the size of the cycle corresponds to $|I|$ being even as required for the lemma. On the other hand, if $|I'|$ is odd, without loss of generality, one may assume that C'_1 contains an

even number of nodes of I' and C'_2 an odd number of them. This implies that $|C'_1| = 4k_1 + 2$ and $|C'_2| = 4k_2$; therefore $|C| = 4(k_1 + k_1 + 1)$, which corresponds to $|I|$ being odd.

As there are no other cases to consider our proof is complete. □

The following lemma can be derived from the previous lemma and the fact that in a 2-factor the number of interior nodes of every cycle must be even (or zero); otherwise, these nodes could not be covered by other cycles.

Lemma 32. *Let $G = (N, L)$ be a solid hexagonal grid network with a 2-factor F. The number of cycles in F is odd if and only if $|N| = 4k + 2$, $k \in \mathbb{N}$.*

Corollary 33. *Let $G = (N, L)$ be a Hamiltonian network. $|N| = 4k + 2$, $k \in \mathbb{N}$.*

Lemma 32 is useful when trying to merge cycles in a 2-factor as to obtain a HC; whereas Corollary 33 obviously provides a necessary condition for a solid hexagonal grid to have a HC. The rest of the necessary conditions provided in this section are based on forbidden configurations. That is, if any of these configurations appear, the network cannot be Hamiltonian. Figure 4.4 illustrates two such configurations: (left) a subgraph that cannot appear in the boundary of the graph, as there is no way to traverse it with a cycle without missing nodes; (right) a generalized shape that cannot be the boundary of the graph, because the cycle would be forced to follow the boundary without visiting interior points.

4.3.2 2-Factor Transformations

Through transforming one 2-factor into another one can possibly obtain a HC. The transformation operation used for this purpose is defined in the following. First we

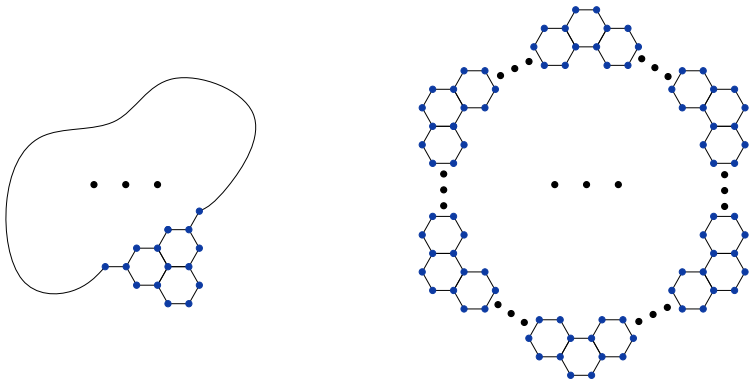

Figure 4.4: Two forbidden configuration for Hamiltonian solid hexagonal grid graphs.

introduce some definitions and basic results. Let $G = (N, L)$ be a solid hexagonal grid and F_1, F_2 2-factors of G. The *symmetric difference* of 2-factors, denoted $F_1 \ominus F_2$, is the set of edges that are in exactly one of the 2-factors (see Figure 4.5 for an example).

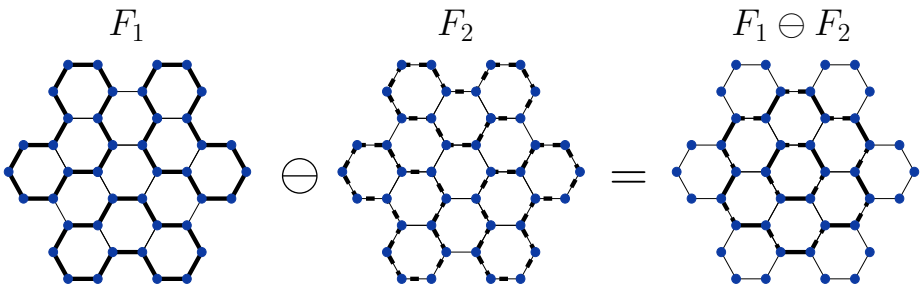

Figure 4.5: Symmetric difference of 2-factors: $F_1 \ominus F_2$.

The fact that the degree of every node in a hexagonal grid is at most three, together with the definition of symmetric difference, leads to the following observation.

Observation 34. *Let $G = (N, L)$ be a solid hexagonal grid and F_1, F_2 2-factors of G. $F_1 \ominus F_2$ consists of a set of non-intersecting cycles, with each cycle consisting of edges that alternate in F_1 and F_2.*

Adopting the terminology from Sheffield [77] we introduce the following definition.

Definition 20. ***(Z-transformation)*** *Let $G = (N, L)$ be a solid hexagonal grid and F_1, F_2 2-factors of G. F_1 and F_2 are said to differ by a* Z-transformation *if $F_1 \ominus F_2$ consists of a cycle of length 6, that is, a hexagon or internal face of G.*

By slight abuse of notation we can say that the application of a Z-transformation to a 2-factor produces another 2-factor. Obviously, for this transformation to be possible the 2-factor must contain a face with three alternating edges, the face where the Z-transformation will be applied. Figure 4.6 shows the application of a Z-transformation on a 2-factor that produces a HC. Although other alternating cycles could be used to transform 2-factors, we will use only Z-transformations as building blocks for our approach to the HC problem.

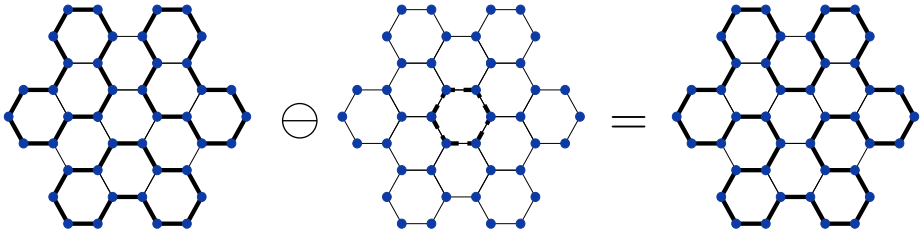

Figure 4.6: Z-transformation applied to a 2-factor that produces a Hamiltonian cycle.

We define a graph $F^2(G)$ whose vertices are the 2-factors of a given graph, G, and the edges consist of pairs of 2-factors that differ by a Z-transformation. The following is a powerful result proved by Sheffield, which we state here in a simplified form as it applies to hexagonal grids.

Theorem 35. *(Adapted from Sheffield [77] Theorem 7) Let $G = (N, L)$ be a bipartite plane network such that every node not on the outer face has constant degree k. Then $F^2(G)$ is connected.*

Notice that Theorem 35 directly entails an algorithm for finding a HC of a graph.

One can simply iterate through all 2-factors in $F^2(G)$ in a depth first search manner by means of Z-transformations until a HC is found. Unfortunately the number of 2-factors of a graph may be exponential and the algorithm is not guaranteed to be asymptotically better than finding a HC by using brute force.

4.3.3 Using Techniques from Square Grids

Our attempt is to mimic, to some extent, the algorithm for constructing HCs in solid square grids of Umans and Lenhart [83]. As for the square grid case, we start with an arbitrary 2-factor of the grid graph and sequentially apply Z-transformations to it with the purpose of reducing the number of cycles in the 2-factor until a HC is found. The existence of such a sequence is supported by Theorem 35, provided that the graph is Hamiltonian.

Umans and Lenhart proved that if a solid square grid G is Hamiltonian, a HC can always be found by reducing the number of cycles in a 2-factor of G one at a time. The equivalent for the case of hexagonal grids is to reduce the number of cycles in the 2-factor by two each time. This is a consequence of Lemma 32, as the number of cycles in a 2-factor of a Hamiltonian solid hexagonal grid must remain odd at all times. In Figure 4.7 we show that depending on the configuration of cycles incident to a hexagon that allows a Z-transformation, the number of cycles in the 2-factor may decrease by two (left), increase by two (middle), or remain the same (right).

It is not always possible to find a single Z-transformation that merges three cycles into one. Therefore, sequences of hexagons need to be considered. The most basic sequence to consider is a *strip* of hexagons. Figure 4.8 shows two example configurations of a 2-factor in which no single Z-transformation reduces the number of

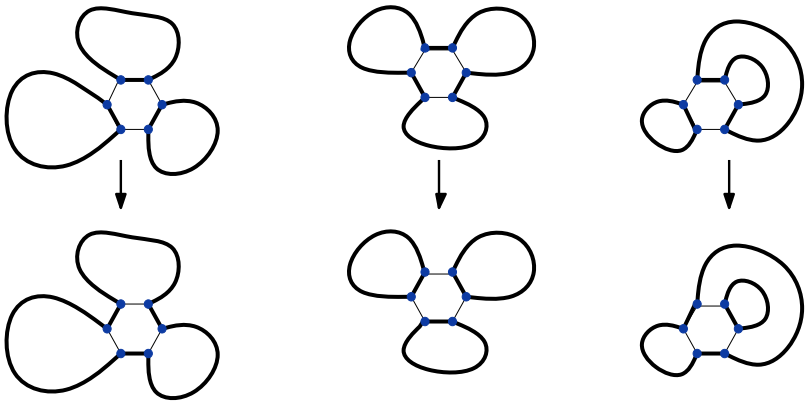

Figure 4.7: Different effects of a Z-transformation on a 2-factor.

cycles; however, after applying Z-transformations to the strip of hexagons (from left to right), the number of cycles decreases. In the example, the strips have lengths two and three. It is noteworthy that the hexagonal grid differs from the square grid, because the length of the strip, or the parity thereof, does not determine whether the number of cycles will decrease or remain the same.

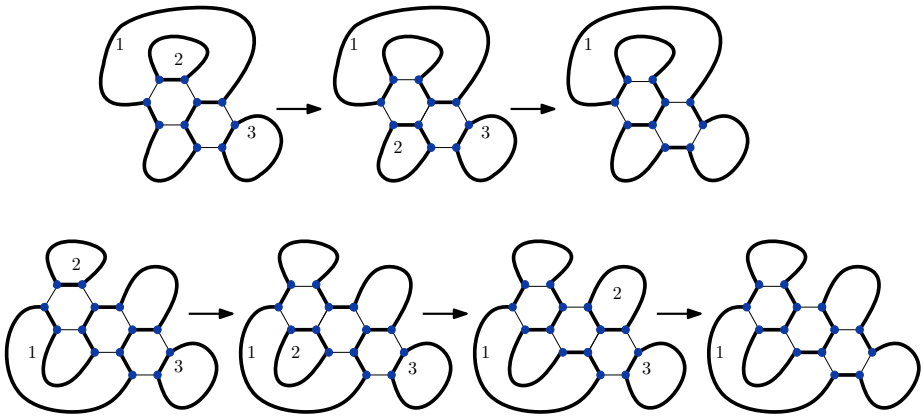

Figure 4.8: Z-transformations along a hexagon strip that reduces the number of cycles.

There is another major difference between the square and hexagonal solid grids when it comes to HCs. Umans and Lenhart [83] proved that there always exists a sequence of strips that reduces the number of cycles in the 2-factor whenever the graph

is Hamiltonian and that such a sequence can be found in polynomial time. Moreover, they proved that any two strips in the sequence do not intersect and can share at most one edge. Unfortunately, the latter does not apply to the hexagonal case; in fact, the number of edges that two strips of hexagons may share is unbounded. Figure 4.9 (left) shows a hexagonal grid for which the only sequence of strips that reduces the number of cycles in the 2-factor contains strips that share multiple edges. (This example can be easily extended to longer strips.) Figure 4.9 (right) also illustrates a similar square grid with the nicer behaviour that strips share at most one edge as proved by Umans and Lenhart. Note that although we have no defined strips or Z-transformations for square grids these concepts can be intuitively carried over from the hexagonal case.

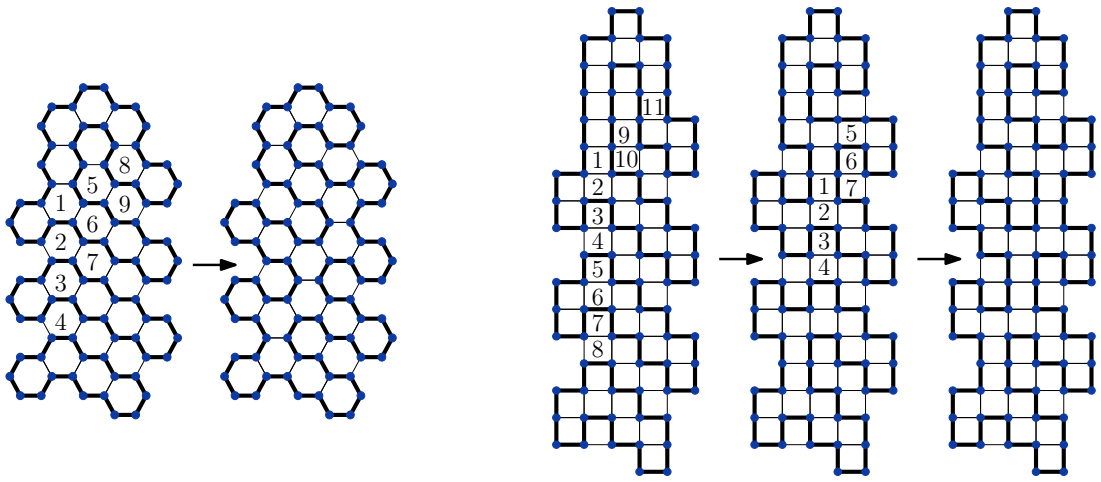

Figure 4.9: Sequences of strips that reduce the number of cycles in a 2-factor.

The previous observation is one reason to believe that the search space for strip sequences is broader for the hexagonal case and, therefore, using this approach the solution is somewhat more involved. Another observation that hints a higher complexity for the HC problem in the hexagonal case is that by inserting a horizontal

diagonal to every hexagon in a solid hexagonal grid, one obtains a graph that is isomorphic to a solid square grid (see Figure 4.10). This suggests that finding a HC in a hexagonal grid is like finding one in the square grid, but using only a subset of the edges.

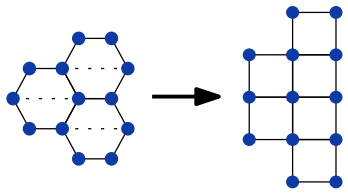

Figure 4.10: Transformation of hexagons into "dominoes" (pairs of vertically arranged squares) and vice verse.

To date the HC problem for solid hexagonal grids remains open. The 2-factor approach possibly leads to a polynomial time algorithm; however, such algorithm seems to call for involved steps and complicated proofs. The apparent lack of simplicity and beauty of a solution of this kind detracts from the theoretical interest of continuing in this direction.

4.4 k-Factors of Graphs

The relevance of 2-factors for the Hamiltonian cycle problem is evident from our previous discussion. In this section we address the more general *k-factor* problem in graphs, as will be defined next. Because the results included herein apply to graphs in general, we will use the terms vertex and edge to refer to the elements of the graph, as opposed to nodes and links. Our results, in a slightly different form, have recently appeared in [49].

In his book "Algorithmic Graph Theory", Gibbons [30] defines a k-factor of a

graph $G = (V, E)$ as a k-regular spanning subgraph of G. According to his definition, a 1-factor is a perfect matching and a 2-factor is a set of disjoint cycles that span G. He also proposes an algorithm to find a 2-factor based on an algorithm that finds a perfect matching of a bipartite graph. The algorithm produces a set of cycles that span the original graph; however, cycles may be *degenerate*, that is, cycles may include the same edge twice. For example, consider $V = \{v_1, v_2, v_3, v_4\}$, $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1)\}$, a 2-factor of G is defined by $F = \{c_1, c_2\}$, where $c_1 = v_1, v_2, v_1$ and $c_2 = v_3, v_4, v_3$. Figure 4.11 shows the problems with Gibbon's algorithm: (left) the figure shows the input graph, (middle) auxiliary bipartite graph where the perfect matching is computed, and (right) resulting 2-factor. Degenerate cycles are marked with double lines. In this example one edge is repeated twice in each cycle. This contradicts the original definition since a subgraph of a simple graph cannot be a multigraph. For the case of $k > 2$, one can also refer to a k-factor, as *degenerate* if it uses one edge multiple times.

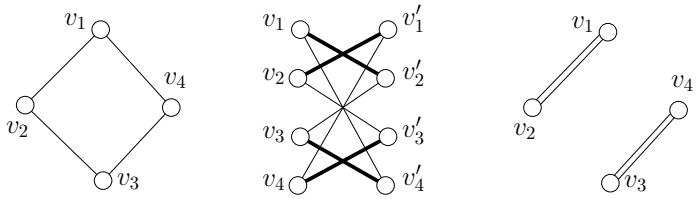

Figure 4.11: Gibbon's algorithm applied to one example graph produces a degenerate 2-factor.

Definition 21. ***(simple k-factor)*** *Consider a graph $G = (V, E)$. A* simple k-factor *of G is a k-factor that is not degenerate.*

According to the previous definition both G and a k-factor of G, F, may include multiple edges (connecting the same pair of vertices) as long as F is a subgraph of

G. However, no edge can be included more than once in F. For clarity, we only refer to simple graphs in the sequel.

Simple k-factors, as opposed to general k-factors, do not only provide a more suitable definition for some problems; they are also more suitable for applications. For example, in Computer Graphics 2-factors are used for fast rendering 3D scenes: a 2-factor of the dual graph of a triangulated scene defines a (partial) linear order of the triangles involved. This allows for efficient *stripification* and compression of large sets of triangles as required for fast rendering [34]. Similarly, 2-factors can be used for stripification of large sets of quadrilaterals and tetrahedra [22]. Also for wireless networks, whenever the graph is not Hamiltonian, 2-factors could be used for the applications presented above (see Section 4.2.2). In all these cases, it is desirable to obtain a small number of long cycles and, preferably, a Hamiltonian cycle. In this regard, degenerate cycles exhibit the worst case, thus demonstrating the preference for simple 2-factors.

The idea of computing 2-factors for obtaining cycles as close as possible to spanning cycles also benefits, in terms of speed, algorithms for computing HCs. In fact, Umans [82] proposed two different algorithms for computing simple 2-factors as part of their algorithms for computing HCs. Their 2-factor algorithms will be reviewed in Section 4.4.2.

In the next section we give an overview of the 1-factor problem (i.e. the perfect matching problem) as it is fundamental for most 2-factor algorithms. In section 4.4.2 we review previous methods for computing 2-factors and simple 2-factors. Our approach to computing simple k-factors is presented in Section 4.4.3.

4.4.1 Maximum Matching Algorithms

The 2-factor algorithm presented by Gibbons [30] and other algorithms that produce simple 2-factors have one thing in common: they rely on maximum matching algorithms.

Our approach to computing k-factors uses the maximum matching algorithm by Micali and Vazirani [51, 67]. Given a graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, their algorithm computes a maximum matching in $O(n^{1/2}m)$ time. This is, however, not the most efficient algorithm for maximum matching in general graphs. Mucha and Sankowski [52] propose a randomized algorithm that runs in $O(n^{\omega})$ time, where $\omega < 2.38$ is the exponent in the complexity of multiplying $n \times n$ matrices. The latter is obviously more efficient if the graph is dense. Other algorithms are also more efficient than Micali and Vazirani's for special types of graphs. For example, Mucha and Sankowski [53] also propose a randomized $O(n^{\omega/2})$ time algorithm for planar graphs. Klein et al. [42] propose an $O(n^{4/3} \log n)$ algorithm for perfect matching in planar bipartite graphs. Biedl et al. [9] present an algorithm for perfect matching in 3-regular bridgeless graphs that runs in $O(n(\log n)^4)$ and $O(n)$ if the graph is also planar. For the non-planar case their algorithm can be improved to $O(n \log^3 n \log \log n)$ time using Thorup's data structure [81]. The reason we use Micali and Vazirani's algorithm is that the graphs to which we apply the maximum matching algorithm, as will be seen in Section 4.4.3, are neither 3-regular, bipartite, planar, nor dense, in general.

4.4.2 Previous Work

The works of Berge [8] and Gabow [27] on k-factor algorithms have been recently brought to our attention. The algorithm presented by Berge is similar to the algorithm

we present in this thesis (see Section 4.4.3) and produces simple k-factors. Like others, this algorithm is based on graphs transformations and computation of a maximum matching on the transformed graphs. By adapting the existing maximum matching algorithms to work more efficiently on the transformed graphs, Gabow improved the complexity of this algorithm, resulting in $O(m\sqrt{kn})$ running time, where n and m are the number of vertices and edges of the input graph, respectively. The algorithm we rediscovered and present in what follows is not as efficient as Gabow's, except for graphs of bounded degree. It remains our main contribution to review other existing algorithms, and point out problems in some of them. Also, we provide two types of transformations as opposed to one, which allows for a more efficient application of the algorithm to the problem instance at hand.

In their surveys, Akiyama and Kano [2], and more recently, Plummer [68], include multiple results describing classes of graphs for which k-factors exist. In some cases an algorithm is also given. One example k-factor algorithm is due to Petersen. Petersen's result establishes the existence of simple $2k$-factors in $2m$-regular graphs, $m \geq k$, and his proof leads to the construction of a simple $2k$-factor.

Gopi and Eppstein [34] propose an algorithm to compute simple 2-factors of 3-regular graphs. Their algorithm computes a perfect matching of the input graph using the algorithm of Biedl et al., as mentioned above. The edges that are not in the computed matching define a simple 2-factor.

Diaz-Gutierrez and Gopi [22] present two different methods to compute simple 2-factors of graphs of maximum degree 4. The *two pass graph matching method* consists of computing a perfect matching on the input graph, removing the edges in the matching from the input graph, and computing a new matching on the remaining

subgraph. The simple 2-factor is defined by the edges in the union of both perfect matchings. The authors acknowledge that their algorithm does not work on graphs with an odd number of vertices even when these are 4-regular. In fact, there are graphs with an even number of vertices where this algorithm also fails. Figure 4.12 shows an example in which this last algorithm fails: (left) the figure shows the input graph, (middle) the perfect matching after the first pass, and (right) a maximum matching that is obtained on the second pass (in thick lines). This suggests that algorithms for computing 2-factors of general graphs need to compute all edges at once, as opposed to using multiple phases.

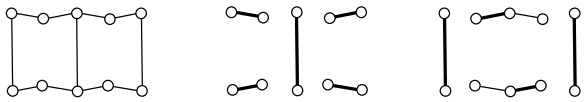

Figure 4.12: Example graph and the results of the two pass matching algorithm.

The second method by Diaz-Gutierrez and Gopi is called the *template substitution algorithm*. In this method, vertices of degree 4 are replaced by *templates*, as shown in Figure 4.13, to obtain an *inflated graph*. A perfect matching of the inflated graph can be translated into a simple 2-factor of the input graph given that exactly two *outside* vertices of a template connect to vertices of other templates. The authors claim that their templates can also replace vertices of degree less than 4; however, no details are provided.

Umans [82] proposes two algorithms for computing simple 2-factors of general graphs. The first algorithm is based on linear programming and, although simple to describe (as a set of linear equations), it is affected by the underlying complexity of linear programming. (See Chvátal's book [16] for details on the complexity of linear programming.) The second approach consists of a set of transformations that produce

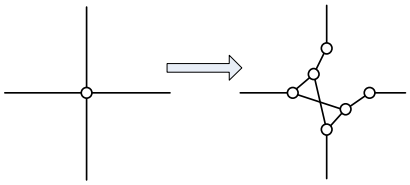

Figure 4.13: Template substitution as in the algorithm by Diaz-Gutierrez and Gopi [22].

a graph similar to the one obtained through the template substitution algorithm (see Figure 4.14 for an example). Thus, a 2-factor of the original graph is found via a perfect matching of the transformed graph.

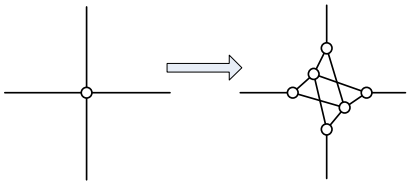

Figure 4.14: Transformation of a degree 4 vertex of the input graph according to Umans' algorithm [82].

4.4.3 An Algorithm for General Simple k-Factors

In this section we present a new algorithm to compute simple k-factors on general graphs. Our technique also relies on matching algorithms. We use a set of *gadgets* that resemble the templates of Diaz-Gutierrez and Gopi and the transformations of Umans, but generalize to vertices of any degree and to k-factors for any possible k value.

The set of gadgets is defined as $\Gamma = \{\Gamma_{2,2}, \Gamma_{2,3}, \ldots, \Gamma_{3,3}, \Gamma_{3,4}, \ldots, \Gamma_{4,4}, \Gamma_{4,5}, \ldots\}$ as shown in Figure 4.15. These gadgets consist of *outer vertices*, *inner vertices*, *core vertices*, *outer edges*, *inner edges*, and *core edges*. A gadget $\Gamma_{k,d}$, $2 \leq k \leq d$,

has d outer vertices, d inner vertices, and k core vertices. Core edges connect core vertices to inner vertices, inner edges connect inner vertices to outer vertices, and outer edges connect outer vertices of different gadget instances. Each inner vertex is adjacent to exactly one outer vertex, while each core vertex is adjacent to all inner vertices. In Figure 4.15 outer vertices are represented by large disks, inner vertices by smaller disks, and core vertices by shaded disks. Vertices within the same class are not adjacent to one another. In our algorithm (see Algorithm 5 below), a vertex v of degree d of the input graph G is replaced by a copy of the $\Gamma_{k,d}$ gadget, v'. The d outer edges of v' are connected to other outer vertices of the gadgets corresponding to the d neighbours of v. As a result, a new graph G' is obtained. We adopt the terminology of Diaz-Gutierrez and Gopi and call G' the *inflated graph* of G. Obviously, a simple k-factor cannot be found in graphs with minimum degree less than k. Thus, we assume that all vertices of G have degree k or more.

Algorithm 5: Simple k-Factor Algorithm

Input: A general graph G
Output: A simple k-factor F, if one exists, or NULL otherwise

Step 1. If G has a vertex of degree less than k, stop and return NULL

Step 2. Build the inflated graph G' from G by the process described above

Step 3. Compute a maximum matching M of G'

Step 4. If M is not a perfect matching, stop and return NULL

Step 5. Initially define F as a graph that contains all the vertices of G and no edges

Step 6. Add all the edges of G that correspond to the outer edges of M to F

Step 7. return F

We provide the following lemma in order to prove that our algorithm correctly

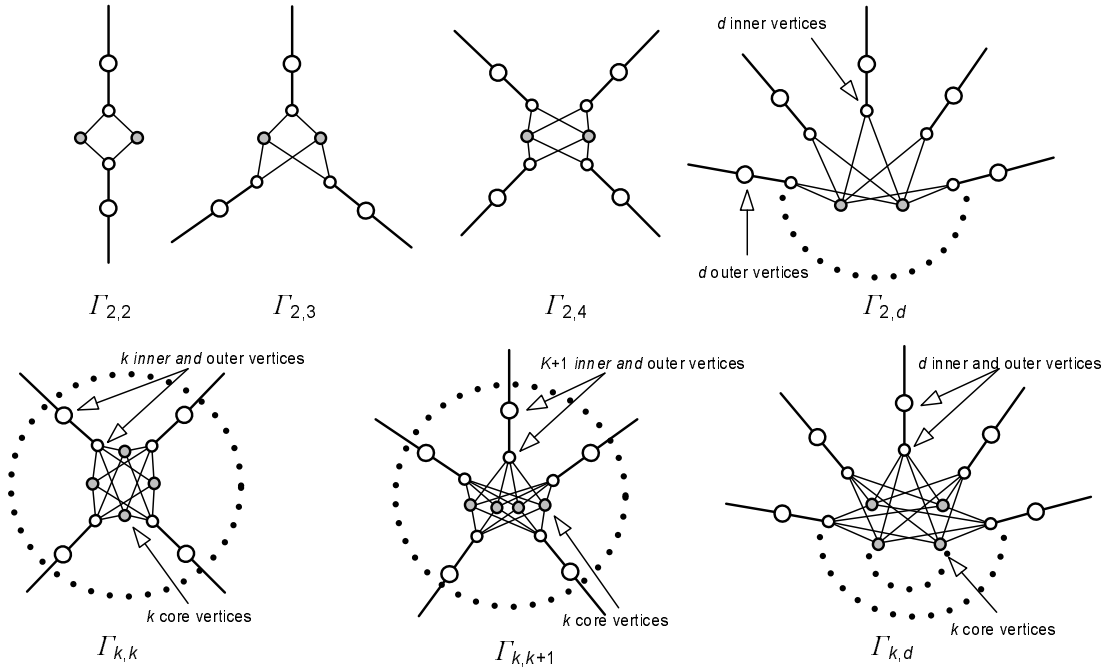

Figure 4.15: Gadgets used in Algorithm 5 to compute a simple k-factor.

computes a simple k-factor of the input graph.

Lemma 36. *Let G be a graph with minimum degree at least k. G has a simple k-factor if and only if the corresponding inflated graph G' has a perfect matching.*

Proof. $\Longrightarrow$ Let F be a simple k-factor of G. Also let v be a vertex of G of degree d and let $v' \subseteq G'$ be the copy of $\Gamma_{k,d}$ that replaces v. Notice that the degree of v in F is k. A perfect matching of G' can be constructed such that k outer edges of v' are part of the matching, those that correspond to the k edges incident to v that are in F. This leaves $d-k$ outer vertices of v' unmatched. The $d-k$ unmatched outer vertices are then matched to their corresponding inner vertices. Finally, the k unmatched inner vertices are matched to the k core vertices, producing a perfect matching for G'.

$\Longleftarrow$ We prove that in a perfect matching M of G' each instance of a gadget $\Gamma_{k,d}$,

$2 \leq k \leq d$, has k outer edges in the matching and, therefore, the corresponding edges in G define a simple k-factor. Recall that the gadgets $\Gamma_{k,d}$, contain k core vertices. The core vertices must be matched in M to k inner vertices. Therefore, the remaining $d-k$ inner vertices must be matched to their adjacent $d-k$ outer vertices, leaving exactly k outer vertices unmatched within v'. These have to be matched with other k vertices of other gadget instances through k distinct outer edges. □

The correctness of Algorithm 5 follows from Lemma 36. We state this explicitly in the following theorem.

Theorem 37. *Let G be a general graph. Algorithm 5 correctly computes a simple k-factor whenever the input graph has one.*

4.4.4 Analysis of the algorithm

The overall performance of our algorithm is based on the performance of the maximum matching algorithm of choice. In general, we use the algorithm by Micali and Vazirani [51, 67]. Let $G = (V, E)$ be the input graph, with $|V| = n$, and $G' = (V', E')$ be the inflated graph obtained from G. Notice that the computation of the maximum matching is the most expensive step of the algorithm. The remaining steps take at most linear time in the number of vertices or edges of the inflated graph. Thus the overall performance of our algorithm is $O(|V'|^{1/2}|E'|)$. In the following we analyze how $|V'|$ and $|E'|$ relate to n in order to express the complexity of the algorithm as a function of the input size.

Since the k-factor problem is easy to solve on complete graphs, we base our analysis on *near-complete* graphs. We define a near-complete graph to be a graph that is not complete but still has $O(n^2)$ edges and the degree of every vertex is $O(n)$. According

to our algorithm, after substituting by Γ gadgets, $|V'|$ and $|E'|$ are $O(n^2)$ and $O(n^2k)$, respectively. This leads to an overall $O(n^3k)$ time complexity.

For graphs with vertex degree bounded by a constant D, the set of Γ gadgets leads to an algorithm that is as efficient as Micali and Vazirani's for maximum matching. In this case, the time complexity follows from $|V'| = O((D + k)n) = O(n)$ and $|E'| = O(Dkn + Dn) = O(n)$, resulting in $O(n^{1.5})$ overall.

When computing simple 2-factors of 3-regular graphs our algorithm produces an inflated graph with vertices of degree 2 and 3. Therefore, Petersen's Theorem (see [9]) does not apply and our algorithm is less efficient than Gopi and Eppstein's algorithm [34]: theirs is $O(n \log^3 n \log \log n)$ while ours is $O(n^{1.5})$.

4.4.5 Gadgets for large k-values

Obviously, the larger the value of k, the more complex the gadgets and therefore the slower the computation of k-factors. However, for large values of k ($k \geq d/2$), a more suitable set of gadgets exists. We define gadget $\Gamma'_{d-k,d}$ as a copy of $\Gamma_{d-k,d}$ that does not contain inner vertices or inner edges. Figure 4.16 illustrates the new gadgets. Only outer and core vertices are needed. Core vertices appear as shaded disks. Core edges directly connect core vertices to outer vertices.

In order to establish that Algorithm 5 also computes a simple k-factor using the $\Gamma'_{d-k,d}$ gadgets, we prove the following lemma.

Lemma 38. *Let G be a graph with minimum degree at least k. G has a simple k-factor if and only if the inflated graph G' obtained by using the $\Gamma'_{d-k,d}$ gadgets has a perfect matching.*

Proof. $\Longrightarrow$ Let F be a simple k-factor of G. Also, let v be a vertex of G of degree d

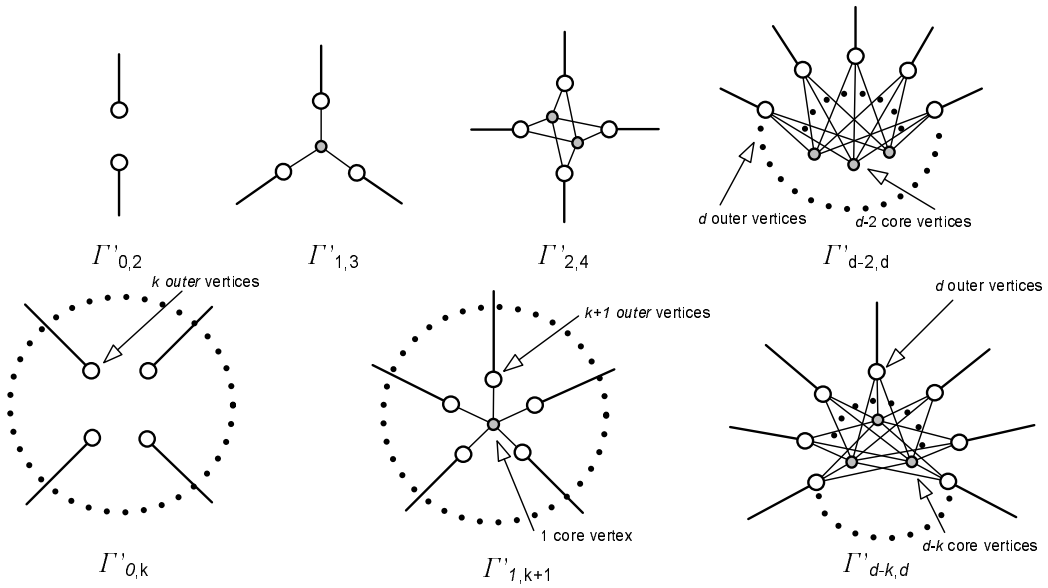

Figure 4.16: Alternate set of gadgets used by Algorithm 5 for large values of k.

and let $v' \subseteq G'$ be the copy of $\Gamma'_{d-k,d}$ that replaces v. Notice that the degree of v in F is k. A perfect matching of G' can be constructed such that k outer edges of v' are part of the matching, those that correspond to the k edges incident to v that are in F. This leaves $d-k$ outer vertices and $d-k$ core vertices of G' unmatched. These are then pairwise matched.

$\Longleftarrow$ We prove that in a perfect matching M of G' each instance of a gadget $\Gamma'_{d-k,d}$, $2 \leq k \leq d$, has k outer edges in the matching and, therefore, the corresponding edges in G define a simple k-factor. Recall that the gadgets $\Gamma'_{d-k,d}$, contain $d-k$ core vertices. The core vertices must be matched in M to $d-k$ outer vertices. Therefore, the remaining k outer vertices must be matched to other k vertices of other gadget instances through k distinct outer edges. □

The advantage of the Γ' gadgets becomes more evident in dense graphs, where vertices have high degrees, and large k-values. We follow with a similar analysis

as presented for the Γ set of gadgets from Section 4.4.4. Let $G = (V, E)$ be the input graph, with $|V| = n$, and $G' = (V', E')$ be the inflated graph obtained from G by using the Γ' gadgets. Assume G is near-complete but $d(v) - k$ is bounded by a constant C for any vertex v, where $d(v)$ denotes the degree of vertex v. Then $|V'| = O(Cn + n^2) = O(n^2)$ and $|E'| = O(Cn^2 + n^2) = O(n^2)$. This results in $O(n^3)$ running time, which is a considerable improvement over the $O(n^3k)$ bound given by the Γ gadgets in this case.

Note that for Algorithm 5 gadgets of the Γ and Γ' sets may be combined. Therefore, the best performance is obtained if the gadget that substitutes a vertex v of degree d of the input graph G is $\Gamma_{k,d}$ for $k < d/2$, or $\Gamma'_{d-k,d}$ otherwise.

4.5 Open Problems

The main subject of study of this chapter remains an open problem itself. This and other related problems of interest are listed in the following.

- Find a polynomial time algorithm to solve the HC poblem for solid hexagonal grids or prove that no such algorithm exists.
- Design an algorithm for computing k-factors with better time complexity than the one proposed by Gabow [27].

4.6 Concluding Remarks

We study the problem of finding Hamiltonian cycles (HC) in solid hexagonal grid graphs. There exist polynomial time algorithms for square and triangular solid grid

graphs, leaving the hexagonal case as the only unsolved regular grid in this respect. We examine an approach to the HC problem that consists of merging cycles of a 2-factor that has been used for solid square grids before. Although the HC problem for solid hexagonal grids remains open, we provide some insight in this direction. Among our positive results are a set of necessary conditions that a solid hexagonal grid must satisfy for it to be Hamiltonian. Also, we present arguments to support our claim that the HC problem for solid hexagonal grids is more complex than for the square grid case.

We also investigate algorithm for computing 2-factors, and more generally, k-factors in graphs. For many applications simple k-factors are more useful than k-factors. We review several algorithms used for computing 2-factors and point out those that do not guarantee to produce simple 2-factors or contain flaws. An algorithm for computing simple k-factors is proposed. The complexity of our algorithm is as optimal for degree bounded graphs. For other cases, the algorithm by Gabow remains the most efficient one [27].

Chapter 5

Summary

We have explored three geometric problems with applications to wireless networks. First, we investigate a recolouring approach that had been previously used for reclassifying geographic data. We solved an open problem on the complexity of geometric recolouring in triangulations and extended the study of this technique to other geometric graphs. Our results in this topic have been recently published in a journal [50]. The same technique was successfully adapted to fault recovery in wireless networks. Our geometric approach to fault recovery has been empirically shown to perform better than a previous combinatorial approach.

Second, we investigated the distributed computation of Voronoi diagrams of networks. None of the previously known non-trivial techniques used for this purpose had been proven to compute the entire Voronoi diagram of a network. Our cooperative approach to computing the Voronoi diagram not only solved this problem efficiently, but was also shown to be simple and independent of the routing mechanisms used by the network. We also introduce a worst-case lower bound and a worst-case efficient algorithm for this problem, in order to create a theoretical framework on this topic.

Our work in this subject builds on previous results related to computing the Voronoi diagram for a special type of network, the unit disk graph [57].

Third, we addressed the problem of finding Hamiltonian cycles in solid hexagonal grid networks. Differently from the previous problems, this one is approached from a centralized computation perspective. Recently, we proved that this problem is NP-hard for general hexagonal grids [37]. The problem remain opens for solid hexagonal grids. We conclude that by following the same approach as for solid square grids, the resulting algorithm would be more involved for the hexagonal case. We derive necessary conditions that help classifying, to some extent, hexagonal grids that may or may not be Hamiltonian. In addition, we propose a polynomial time algorithm for computing 2-factors that can be used for finding a Hamiltonian cycle. Our algorithm generalizes to k-factors and guarantees that the output is a simple graph. Our results on simple k-factors, as appeared in [49], are presented in a more general graph theoretical context, because they are of independent interest to this field.

At the end of each chapter we include open problems and remarks regarding each one of these topics.

Bibliography

[1] M. Adler, E.D. Demaine, N.J.A. Harvey, and M. Pătraşcu. Lower bounds for asymmetric communication channels and distributed source coding. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 251–260, New York, NY, USA, 2006. ACM.

[2] J. Akiyama and M. Kano. *Book of Factors and Factorizations of Graphs*. 2007. Online version: http://gorogoro.cis.ibaraki.ac.jp/web/papers/FactorGraphVer1A4.pdf.

[3] K. Alzoubi, X.Y. Li, Y. Wang, P.J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.

[4] E.M. Arkin, S.P. Fekete, K. Islam, H. Meijer, J.S.B. Mitchell, Y. Núñez-Rodríguez, V. Polishchuk, D. Rappaport, and H. Xiao. Not being (super)thin or solid is hard: A study of grid hamiltonicity. *Computational Geometry: Theory and Applications*, 42(6-7):582–605, 2009.

[5] E.M. Arkin, S. Khuller, and J.S.B. Mitchell. Geometric knapsack problems. *Algorithmica*, 10:399–427, 1993.

[6] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[7] B.A. Bash and P.J. Desnoyers. Exact distributed voronoi cell computation in sensor networks. In *Proc. 6th International Conference on Information Processing in Sensor Networks (IPSN'07)*, pages 236–243, 2007.

[8] C. Berge. *Graphs and Hypergraphs*. North-Holland, 1973.

[9] T.C. Biedl, P. Bose, E.D. Demaine, and A. Lubiw. Efficient algorithms for petersen's matching theorem. *Journal of Algorithms*, 38(1):110–134, 2001.

[10] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7:609–616, 2001.

[11] A. Boukerche and K. Abrougui. An efficient leader election protocol for mobile networks. In *Proc. 2006 International Conference on Wireless Communications and Mobile Computing (IWCMC'06)*, pages 1129–1134, 2006.

[12] J. Bruck, J. Gao, and A. Jiang. Localization and routing in sensor networks by local angle information. *ACM Transactions on Sensor Networks*, 5(1):1–31, 2009.

[13] I. Caragiannis, C. Kaklamanis, E. Kranakis, D. Krizanc, and A. Wiese. Communication in wireless networks with directional antennas. In *Proc. 20th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA'08)*, pages 344–351, 2008.

[14] T.M. Chan. Low-dimensional linear programming with violations. In *43rd IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 570–579, 2002.

[15] W.P. Chen, J.C. Hou, and L. Sha. Dynamic clustering for acoustic target tracking in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 3(3):258–271, 2004.

[16] V. Chvátal. *Linear Programming*. W. H. Freeman, San Francisco CA, 1983.

[17] I. Cidon and O. Mokryn. Propagation and leader election in a multihop broadcast environment. In *Proc. 12th International Symposium on Distributed Computing (DISC'98)*, pages 104–118, 1998.

[18] B.N. Clark, C.J. Colbourn, and D.S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.

[19] A. Clarridge. Cellular automata: Algorithms and applications. Master's thesis, School of Computing, Queen's University, 2009.

[20] S.A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd Symposium on Theory of Computing (STOC)*, pages 151–158. ACM, New York, 1971.

[21] Y. de la Noval, F. Luccio, Y. Núñez-Rodríguez, and L. Pagli. Dynamos in three-dimensional meshes. In *Proc. 7th. International Congress on Computer Science Research*, pages 51–62, 2000.

[22] P. Diaz-Gutierrez and M. Gopi. Quadrilateral and tetrahedral mesh stripification using 2-factor partitioning of the dual graph. *The Visual Computer*, 21(8-10):689–697, 2005.

[23] P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recognition Letters*, 14:715–718, 1993.

[24] R.B. Ellis, X. Jia, and C. Yan. On random points in the unit disk. *Random Structures and Algorithms*, 29(1):14–25, 2006.

[25] P. Flocchini, E. Lodi, F. Luccio, L. Pagli, and N. Santoro. Dynamic monopolies in tori. *Discrete Applied Mathematics*, 137(2):197–212, 2004.

[26] S. Fortune. A sweepline algorithm for voronoi diagrams. In *Proc. 2nd Annual Symposium on Computational Geometry (SCG'86)*, pages 313–322, 1986.

[27] H.N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proc. 15th Annual ACM Symposium on Theory of Computing*, pages 448–456, 1983.

[28] R.G. Gallager, P.A. Humblet, and P.M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.

[29] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[30] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[31] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[32] E. Goles and J. Olivos. Periodic behaviour of generalized threshold functions. *Discrete Mathematics*, 30:187–189, 1980.

[33] J.E. Goodman and editors J. O'Rourke. *Handbook of Discrete and Computational Geometry*. CRC Press, 2nd edition, 2004.

[34] M. Gopi and D. Eppstein. Single-strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum (EUROGRAPHICS)*, 23(3):371–379, 2004.

[35] J.L. Hill. *System Architecture for Wireless Sensor Networks*. PhD thesis, University of California, Berkeley, 2003.

[36] L.H. Hsu and C.K. Lin. *Graph Theory and Interconnection Networks*. CRC Press, 2008.

[37] K. Islam, H. Meijer, Y. Núñez-Rodríguez, D. Rappaport, and H. Xiao. Hamiltonian circuits in hexagonal grid graphs. In *Proc. 19th Canadian Conference on Computational Geometry (CCCG'07)*, pages 85–88, 2007.

[38] A. Itai, C.H. Papadimitriou, and J.L Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[39] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane: a survey. *Discrete and Computational Geometry*, 25:551–570, 2003. The Goodman-Pollack Festschrift, Algorithms and Combinatorics.

[40] B. Karp and H.T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proc. 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, pages 243–254, 2000.

[41] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. In

Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08), pages 263–272, 2008.

[42] P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC'94)*, pages 27–37. ACM Press, 1994.

[43] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry (CCCG'99)*, pages 51–54, 1999.

[44] B. Krishnamachari and S. Iyengar. Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *IEEE Transactions on Computers*, 53(3):241–250, 2004.

[45] D.T. Lee and F.P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal of Computation*, 6(3):594–606, 1977.

[46] J. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with delaunay triangulation overlays. Technical Report CS-2001-26, University of Virginia, Department of Computer Science, 2001.

[47] F. Luccio, L. Pagli, and H. Sanossian. Irreversible dynamos in butterflies. In *Proc. 6th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 204–218. Carleton University Press, 1999.

[48] X. Luo, M. Dong, and Y. Huang. On distributed fault-tolerant detection in wireless sensor networks. *IEEE Transactions on Computers*, 55(1):58–70, 2006.

[49] H. Meijer, Y. Núñez-Rodríguez, and D. Rappaport. An algorithm for computing simple k-factors. *Information Processing Letters*, 109(12):620–625, 2009.

[50] H. Meijer, Y. Núñez-Rodríguez, and D. Rappaport. Bounds for point recolouring in geometric graphs. *Computational Geometry: Theory and Applications*, 42(6-7):690–703, 2009.

[51] S. Micali and V. V. Vazirani. An $o(\sqrt{|V|}|e|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21st Annual Symp. on Foundations of Computer Science (FOCS '80)*, pages 17–27, 1980.

[52] M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *Proc. 45th Annual Symp. on Foundations of Computer Science (FOCS '04)*, pages 248–255, 2004.

[53] M. Mucha and P. Sankowski. Maximum matchings in planar graphs via gaussian elimination. *Algorithmica*, 45(1):3–20, 2006.

[54] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[55] D. Niculescu and B. Nath. Ad hoc positioning system (aps) using aoa. In *Proc. of 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, volume 3, pages 1734–1743, 2003.

[56] F.B. Nocetti, I. Stojmenović, and J. Zhang. Addressing and routing in hexagonal networks with applications for tracking mobile users and connection rerouting in cellular networks. *IEEE Transactions on Parallel Distributed Systems*, 13(9):963–971, 2002.

[57] Y. Núñez-Rodríguez, H. Xiao, K. Islam, and W. Alsalih. A distributed algorithm for computing voronoi diagrams in the unit disk graph model. In *Proc. 20th Canadian Conference in Computational Geometry (CCCG'08)*, pages 199–202, 2008.

[58] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley, 2nd edition, 2000.

[59] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998.

[60] J. Pach, editor. *Towards a Theory of Geometric Graphs*. American Mathematical Society, 2004.

[61] P. Parvathipuram, V. Kumar, and G.C. Yang. An efficient leader election algorithm for mobile ad hoc networks. In *Proc. International Conference on Distributed Computing and Internet Technology (ICDCIT'04)*, pages 32–41, 2004.

[62] S. Patil, S.R. Das, and A. Nasipuri. Serial data fusion using space-filling curves in wireless sensor networks. In *Proc. 1st IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON'04)*, pages 182–190, 2004.

[63] D. Peleg. Time-optimal leader election in general networks. *Journal of Parallel and Distributed Computing*, 8(1):96–99, 1990.

[64] D. Peleg. Local majority voting, small coalitions and controlling monopolies in graphs: A review. Technical Report CS96-12, Department Of Mathematics & Computer Science, Weizmann Institute Of Science, 1996. Online version: http://wisdomarchive.wisdom.weizmann.ac.il/archive/00000025.

[65] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[66] M. Penrose. *Random Geometric Graphs*. Oxford University Press, 2004.

[67] P.A. Peterson and M.C. Loui. The general maximum matching algorithm of micali and vazirani. *Algorithmica*, 3(1):511–533, 1988.

[68] M.D. Plummer. Graph factors and factorization: 1985-2003: a survey. *Discrete Mathematics*, 307:791–821, 2007.

[69] V. Polishchuk, E.M. Arkin, and J.S.B. Mitchell. Hamiltonian cycles in triangular grids. In *Proc. 18th Canadian Conference on Computational Geometry (CCCG'06)*, pages 63–66, 2006.

[70] K.S. Prabh and T.F. Abdelzaher. On scheduling and real-time capacity of hexagonal wireless sensor networks. In *Proc. 19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, pages 136–145, 2007.

[71] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proc. 6th Annual International Conference on Mobile Computing and Networking (MOBICOM'00)*, pages 32–43, 2000.

[72] I. Reinbacher, M. Benkert, M. van Kreveld, J.S.B. Mitchell, J. Snoeyink, and A. Wolff. Delineating boundaries for imprecise regions. *Algorithmica*, 50(3):386–414, 2008.

[73] S.K. Sarkar, T.G. Basavaraju, and C. Puttamadappa. *Ad Hoc Mobile Wireless Networks: Principles, Protocols, and Applications*. Auerbach Publications, Taylor & Francis Group, 2008.

[74] C. Seara. *On Geometric Separability.* PhD thesis, Universitat Politècnica De Catalunya, 2002.

[75] M.I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annual Symposium on Foundations of Computer Science (SFCS'75)*, pages 151–162, 1975.

[76] M. Sharifzadeh and C. Shahabi. Supporting spatial aggregation in sensor network databases. In *Proc. 12th Annual ACM International Workshop on Geographic Information Systems (GIS'04)*, pages 166–175, 2004.

[77] S. Sheffield. Computing and sampling restricted vertex degree subgraphs and hamiltonian cycles, 2001. Online version: http://www.citebase.org/abstract?id=oai:arXiv.org:math/0008231.

[78] D. Slepian and J.K. Wolf. Noiseless encodings of correlated information sources. *IEEE Transactions on Information Theory*, 19(4):471–480, 1973.

[79] A.A. Somasundara, A. Ramamoorthy, and M.B. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Proc. 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 296–305, 2004.

[80] G. Takahara, K. Xu, and H.S. Hassanein. Efficient coverage planning for grid-based wireless sensor networks. In *Proc. IEEE International Conference on Communications (ICC 2007)*, pages 3522–3526, 2007.

[81] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd Annual ACM Symposium on Theory of Computing*, pages 343–350, 2000.

[82] C. Umans. An algorithm for finding hamiltonian cycles in grid graphs without holes. Honor Thesis, 1996. Williams College.

[83] C. Umans and W. Lenhart. Hamiltonian cycles in solid grid graphs. In *Proc. 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*, pages 496–507, 1997.

[84] S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Proc. 12th IEEE International Conference on Network Protocols (ICNP'04)*, pages 350–360, 2004.

[85] J. Walrand and P. Varaiya. *High-Performance Communication Networks*. Morgan Kaufmann Publishers Inc., 2nd edition, 2000.

[86] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proc. 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 381–396, 2006.

[87] F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann Publications, Elsevier, 2004.

[88] Z. Zhou, S.R. Das, and H. Gupta. Variable radii connected sensor cover in sensor networks. *ACM Transactions on Sensor Networks*, 5(1):1–36, 2009.

Glossary

Appendix A

Infinite Recolouring

A.1 Infinite Recolouring Sequences

Figures A.1 and A.2 show different stages of the recolouring sequence of the plane network shown in Figure 2.6. Nodes represented by small disks never change colour. Recoloured nodes are pointed out by arrows. The four recolourings of each stage can occur in any order, one at a time. Notice that the colouring on the last stage (see stage 8 in Figure A.2) resembles a 90°-rotation of the initial colouring. By concatenation of similar recolouring sequences the colours can continue to rotate an infinite number of times.

Figure A.3 shows a non-plane network with an infinite recolouring sequence that includes all the nodes. Recolourings occur in the order indicated by the numbers in the figure. Notice that the figure on the right is a rotation of the figure on the left.

Figure A.4 shows a 1-bend embedding of a wired network with an infinite recolouring sequence that includes all the nodes. This example also shows an order in which the nodes can be recoloured such that colours rotate an infinite number of times.

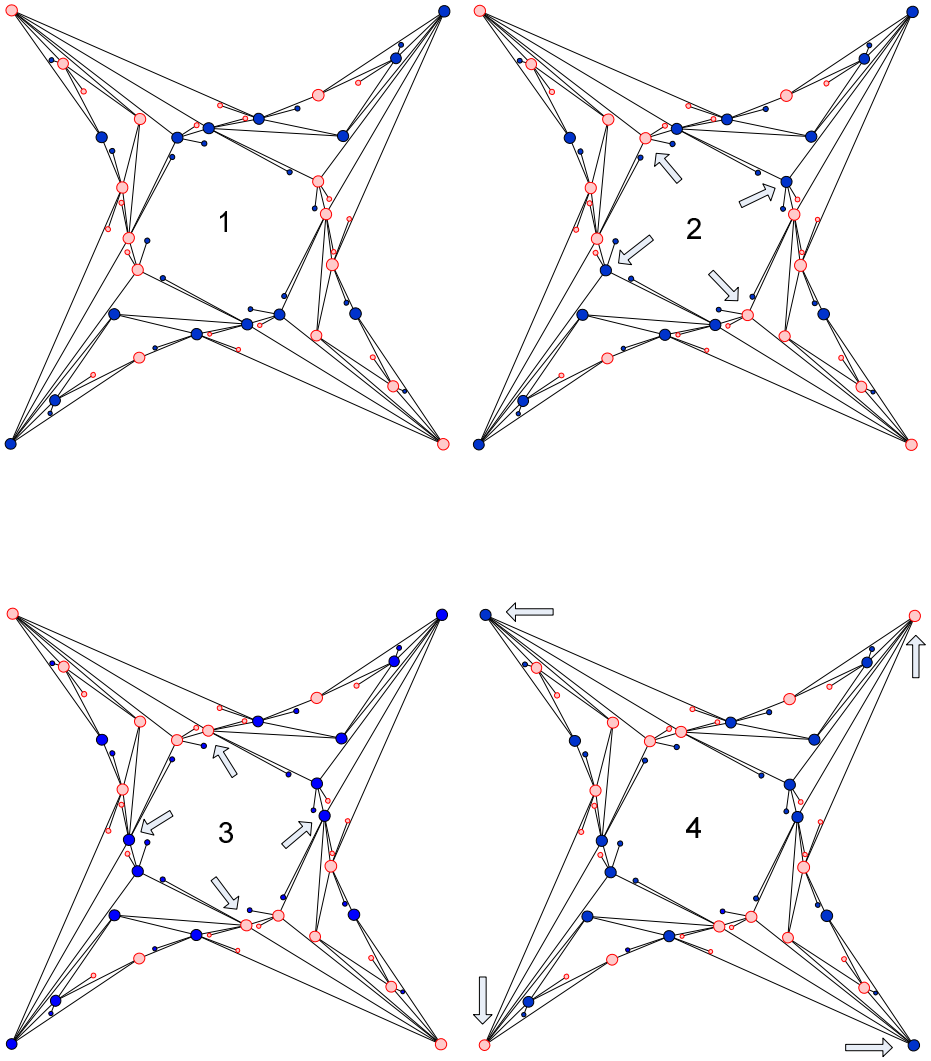

Figure A.1: Infinite recolouring sequence of a plane network (Part I).

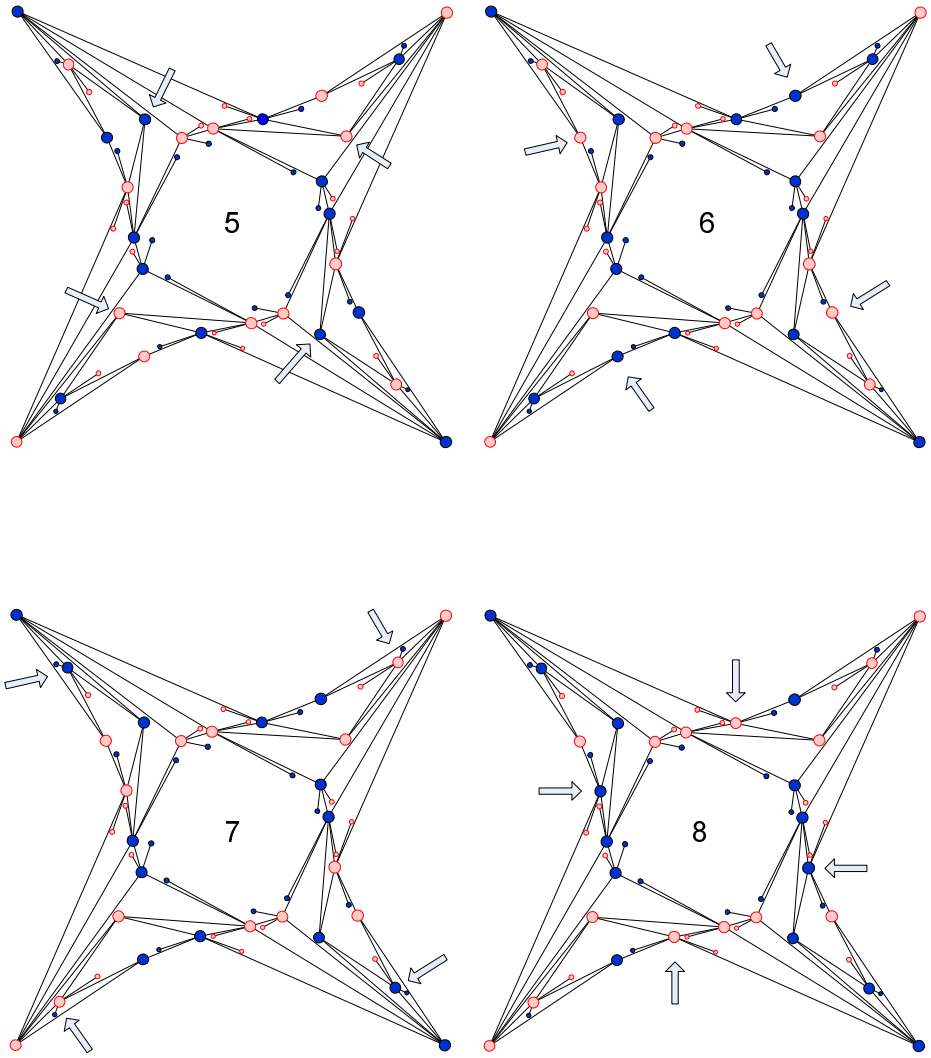

Figure A.2: Infinite recolouring sequence of a plane network (Part II).

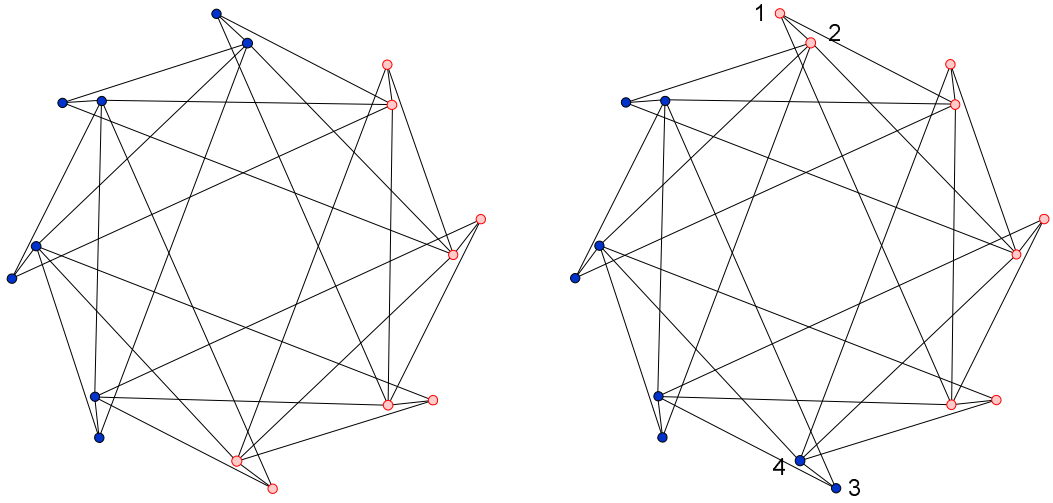

Figure A.3: Infinite recolouring sequence of a non-plane network.

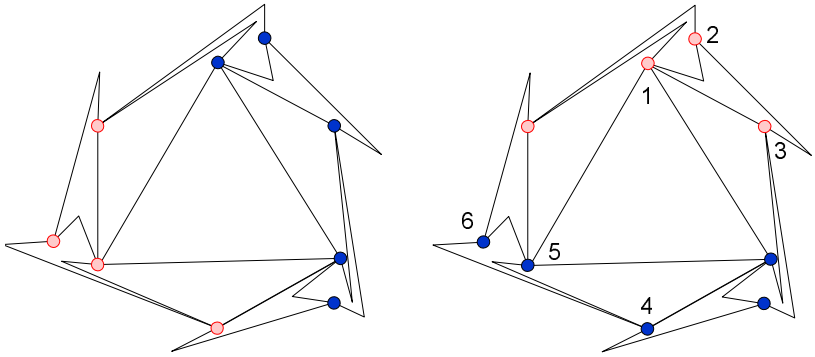

Figure A.4: Infinite recolouring sequence of a plane embedding of a network with 1-bend links.